

Univerzita Karlova
Pedagogická Fakulta

Katedra informačních technologií a technické výchovy

BAKALÁŘSKÁ PRÁCE

Ověření multiplatformnosti Windows 10 na počítači
Raspberry Pi

Verification of multiplatformity of Windows 10 on
computer Raspberry Pi

Petrásek Jan

Vedoucí práce: PhDr. Martin Stejskal

Školitel za JU: Mgr. Jiří Pech, PhDr.

Studijní program: Specializace v pedagogice

Studijní obor: B IT (7507R040)

Odevzdáním této bakalářské práce na téma „Ověření multiplatformnosti Windows 10 na počítači Raspberry Pi“ potvrzuji, že jsem ji vypracoval pod vedením vedoucího práce samostatně za použití v práci uvedených pramenů a literatury. Dále potvrzuji, že tato práce nebyla využita k získání jiného nebo stejného titulu.

České Budějovice 20.11.2021

Poděkování

Tímto bych rád poděkoval školiteli Mgr. Jiřímu Pechovi Ph.D. za odbornou pomoc při zpracování mé bakalářské práce v roce 2016 pod UAI PŘF JU.

Dále bych rád poděkoval Martinu Stejskalovi, že se ujal vedení aktualizace práce za Univerzitu Karlovu, Leoši Dvořákovi a Petru Žilavému, že za KDF MFF umožnili realizovat potřebná měření a tím včasné odevzdání práce.

Také chci poděkovat Janu Pospíšilovi (Technickému evangelistovi Microsoftu ČR) za odborné poradenství v oblasti operačního systému a psaní aplikací pro tento systém a Michaelu Grafnetterovi (nezávislému IT konzultantovi pracujícím pro Microsoft, Gopas a Univerzitu Karlovu) za odborné konzultace v oblasti Windows PowerShell.

Velký dík patří také Heidemarii Wiesnerové, která pomáhala s překlady oficiálních zdrojů z anglického jazyka.

V neposlední řadě chci také poděkovat své manželce za aktivní podporu při dokončování i aktualizace práce.

Bibliografické údaje

Petrásek, J., 2016: Ověření multiplatformnosti Windows 10 na počítači Raspberry Pi2. [Verification of multiplatformity of Windows 10 on computer Raspberry Pi2. Bc. Thesis, in Czech.] -85p, Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Abstrakt

Bakalářská práce se týká Windows 10 IoT Core a ověření jejich funkčnosti na platformě Raspberry Pi. Je ověřen postup instalace systému a jeho prvotní konfigurace. Je popsána problematika řízení systému pomocí Webserveru a PowerShellu 5. Pro ověření vlastností systému jsem využil testy práce s GPIO sběrnici a testy simulují reálné IoT scénáře.

Klíčová slova

Windows 10, IoT, Raspberry Pi, Visual Studio 2015, Windows Universal Apps.

Abstract

This bachelor thesis concerns the Windows 10 IoT Core and verification of its functionality on the platform Raspberry Pi. The process of installation and initial configuration is validated. The system management using PowerShell 5 or webserver is described. To verify the properties of operating system I tested the performance using GPIO bus and tests simulating realistic IoT scenarios.

Key words

Windows 10, IoT, the Raspberry Pi, Visual Studio 2015, Windows Universal Apps.

Obsah

ÚVOD	9
1. CÍLE A METODIKA PRÁCE	11
1.1. CÍLE PRÁCE	11
1.2. METODIKA PRÁCE.....	11
2. Nasazení Windows 10 IoT Core na Raspberry Pi2	12
2.1. O systémech Windows 10 IoT	12
2.2. Požadavky Windows 10 IoT Core	13
2.3. Získání systému.....	15
2.3.1. Užitý systém pro práci.....	17
2.4. Specifika Windows 10 IoT Core	17
2.5. Instalace systému Windows 10 IoT Core.....	18
2.5.1. Základní konfigurace Windows 10 IoT Core.....	21
2.5.2. Webové rozhraní pro řízení systému.....	22
2.6. Práce se systémem pomocí PowerShellu	26
2.6.1. Procesy	26
2.6.2. Připojení k síti	28
2.6.3. Výpis informací o procesoru	29
2.6.4. Správa uživatelských účtů	29
2.6.5. Další praktické příkazy.....	29
2.6.6. Práce s registry OS	30
2.6.7. Konfigurace služeb.....	31
2.6.8. Úprava spouštění OS	31
2.7. Aplikace pro Windows 10 IoT Core	31
2.7.1. Vzdálené ladění kódu	36
2.7.2. Vytvoření balíčku aplikace.....	38
2.7.3. Přímá instalace pomocí Visual Studia.....	41

2.7.4. Aplikace v jazyce Python	42
2.7.5. Ladění a nasazení UWP v C++	43
2.8. Srovnání vývoje mezi verzemi 10531 a 14342	43
2.8.1. Novinky v řídicích aplikacích pro stolní počítač.....	43
2.8.2. Novinky v instalaci systému	44
2.8.3. Novinky v systému pro build 10.0.14342.1000	45
3. Testování systému	47
3.1. Strategie testování	47
3.1.1. Sledované parametry	49
3.2. Metodika testování	49
3.2.1 Test aplikace IoT scénářů.....	49
3.2.2. Test práce s internetem.....	50
3.2.3. Test reakční doby na akci GPIO.....	50
3.2.4. Test přesnosti SW časovačů	52
3.3. Výsledky testů	53
3.3.1. Test aplikace IoT scénářů.....	53
3.3.2. Test práce s internetem.....	56
3.3.3. Test reakční doby na akci GPIO.....	58
3.3.4. Test přesnosti SW časovačů	61
4. Ukončení podpory	63
4.1. Odkaz Windows 10 IoT Core.....	64
4.2. Co dál s projekty pro Windows 10 IoT Core?	65
4.2.1. Vzdal Microsoft multiplatformnost a domácí IoT?.....	65
5. Projekt WoR.....	66
5.1. Nasazení Windows dle WoR.....	66
5.1.1. Potřebný HW	67
5.1.2. Potřebné úpravy konfigurace UFI	68
5.1.3. Doporučená úprava konfigurace Windows po prvním spuštění	68

5.2. Aplikační kompatibilita.....	69
5.3. Výkonové srovnání Windows on ARM a Windows x64.....	70
5.3.1. Konfigurace zařízení	71
5.3.2. Subjektivní srovnání na Raspberry Pi 4	71
5.4. Spuštění UWP využívajících GPIO	72
5.5. Srovnání Windows 10 on ARM s Windows 10 IoT Core a Raspberry Pi OS při práci s GPIO	74
5.6. Hodnocení projektu WoR.....	78
Závěr – Výsledné hodnocení Windows 10 IoT Core	81
Pohled z roku 2021	83
Seznam použitých obrázků.....	85
Seznam Tabulek	88
Seznam grafů.....	89
Seznam použité literatury	85
Publikace	85
Internetové zdroje.....	85
Seznam použitých zkratk.....	90
Seznam příloh.....	92

ÚVOD

Téma své bakalářské práce jsem si zvolil na základě zkušeností s testováním Windows 10 pro domácí počítače a zajímalo mě, jestli se Microsoftu podaří s Windows proniknout i na zařízení typu Raspberry. Jednodeskové počítače jako je Raspberry Pi2 nebo MinnowBoard Max jsou v dnešní době velmi oblíbené pro své velké možnosti, což si uvědomují i v Microsoftu. Do příchodu Windows 10 IoT ovládaly tento sektor různé Linuxové distribuce. Na jednoúčelových zařízeních jako jsou bankomaty, či pokladní a informační systémy se snažil Microsoft držet krok pomocí systému řady embedded, od roku 2015 označovaných jako IoT Enterprise. Tato řada nikdy nebyla široce dostupná vývojářům a bylo nutné objednávat hodně licencí, které stály od 100 dolarů výše. Současný název lépe vystihuje, že místem určení těchto systémů je korporátní sektor a jeho produkty.

Rodina systémů Windows 10 představuje první opravdu multiplatformní operační systém z dílen Microsoftu, ale tomu tak? Když se podíváme na vývoj systémů Windows, tak jejich hlavní doménou jsou osobní počítače a servery. Na osobních počítačích je již k dispozici poslední verze Windows 10, a finální verze Windows 11, která je průběžně vylepšována a pro servery vyšla finální verze na konci září 2016, respektive v listopadu 2018. Mezi mobilními telefony se od verze Windows Phone 8.1 podařilo Microsoftu systém rozšířit i mimo modelovou řadu Lumia a vybrané modely HTC, a i zde je již k dispozici jak vývojová, tak finální verze systému Windows 10 Mobile, která se postupně připravuje pro jednotlivé podporované modely telefonů. S koncem roku 2019 bohužel Microsoft rozhodl o ukončení mobilní platformy a nahradil ji Androidem. Novinkou je však verze pro IoT, kde Microsoft sice není úplně poprvé, ale jeho Embedded verze Windows se příliš nerozšířily. Odpověď na otázku, zda se Microsoftu opravdu podařilo přijít s konkurenci schopným IoT systémem a dosáhnout tak plné multiplatformnosti rodiny Windows 10 jsem se pokusil zodpovědět v této práci.

V první části práce je vytvořena podrobná instalační metodika systému. Jsou zde uspořádány informace z různých zdrojů Microsoftu doplněné o mé vlastní postřehy. Jsou zde uvedeny všechny SW a HW požadavky pro instalaci systému i pro jeho běh. Celá instalace a konfigurace systému je popsána podrobně krok za krokem, stejně jako využití nástrojů pro vzdálené ladění ve Visual studiu i instalace aplikací na tento systém. V tomto bloku také udám přesnou specifikaci systému a HW vybavení, na kterém probíhali testy systému.

Je to z toho důvodu, že některé výsledky testů se mohou s využitím jiného obrazu systému výrazně lišit.

V druhé části jsou popsány vlastní testy provedené na systému a jejich vyhodnocení. Součástí testování je i nasazení systému v některých vzorových příkladech (tzv. Hands on labs) a vyhodnocení provedení požadovaných úkonů. Protože jde o IoT systém, který komunikuje s externím HW, jako jsou snímače, převodníky apod. pomocí GPIO sběrnice a I²C sběrnice, tak je do testování systému též zařazen i test, který zkoumá odezvu GPIO sběrnice z pohledu externího zařízení. Závěrečné vyhodnocení testů je inspirováno modelem kvality SW FURPS od společnosti HP, o kterém si myslím, že vystihuje všechno potřebné pro účely této práce.

Na konci této práce se zamýšlím nad projektem WoR, který umožňuje využít Raspberry Pi jako alternativu k běžnému PC, ale především poskytuje určitou možnost běhu projektů, které vznikly pro Windows 10 IoT Core po konci jejich podpory.

1. CÍLE A METODIKA PRÁCE

1.1. CÍLE PRÁCE

Jedním z cílů mé bakalářské práce je na základě praktické zkušenosti s používáním systému a studia oficiálních zdrojů vytvořit ucelený zdroj informací, který poskytne uživateli vše potřebné, co potřebuje vědět pro nasazení systému Windows 10 IoT Core ve svém projektu. Jde o požadavky systému, jeho charakteristické rysy, ale především o postupy umožňující jeho reálné nasazení a následnou správu.

Druhým z cílů je na testech prokázat jak kvality systému, tak jeho nedostatky. Hlavním úkolem praktické části je názorně ukázat, kdy je Windows 10 IoT dobrou volbou pro nasazení na projektu a kdy by jeho volba mohla s sebou přinést jisté komplikace. Ze získaných poznatků a dat následně vytvořit souhrnné resumé o vlastnostech systému.

Posledním podbodem je aktualizace dat na Pi4 v roce 2021 s aktuálně podporovanou alternativou z rodiny Windows 10.

1.2. METODIKA PRÁCE

První část práce je vypracována na základě studia oficiálních informací na webu Microsoftu a studia záznamu z konference „Otevřené Widle“. Následně jsou jednotlivé kroky ověřeny a podrobně popsány. Pro větší názornost jsou součástí popisů screen's jednotlivých kroků a vlastní postřehy. Po dokončení instalace a konfigurace na jednoduchém příkladu ukáží Windows Universal Apps, jak ji ladit a následně nainstalovat na Windows 10 IoT Core.

Druhá část je vedena na základě studia odborné literatury z oblasti testování SW, podkladů z certifikačního školení řízení jakosti SW a zkušeností s testováním několika generací systémů Windows pro stolní PC. Postupným provedením jednotlivých testů a jejich vyhodnocením je získána objektivní představa o vlastnostech systému.

2. Nasazení Windows 10 IoT Core na Raspberry Pi2

2.1. O systémech Windows 10 IoT

„Windows 10 IoT patří do rodiny Windows 10, která je zaměřená na široký okruh inteligentních přístrojů, od malých průmyslových bran k větším, komplexnějším přístrojům jako jsou místa prodejních terminálů a ATMs. Kombinace nejnovějšího vývoje nástrojů Microsoftu a partnerů služeb Azure IoT umožňuje shromažďovat, ukládat a zpracovávat data, vytvářet obchodní inteligenci a zefektivnit obchodní výnosy. Řešení založené na této spolupráci, jenž se opírá o Windows 10 IoT, zajistí rozšiřující se možnosti s využitím plně šíře technologií Microsoftu tak, aby byla nabídnuta řešení na principu „end-to-end“.

Windows 10 IoT Enterprise přináší schopnosti Windows 10 Enterprise do širokého spektra průmyslových přístrojů od maloobchodu, přes výrobu, oblast zdraví, financí až k dalším průmyslovým odvětvím. Přístroje Windows 10 IoT Enterprise řídí řadu obchodních aplikací a vykonávají specializované funkce bezpečným, spolehlivým a aerodynamickým způsobem tak, aby podpořili mise rozhodujících průmyslových přístrojů. Windows 10 IoT Enterprise podporuje jak aplikace Universal Windows, tak Classic Windows a mnoho dalších inovativních schopností: pokročilá ochrana proti moderním bezpečnostním hrozbám, plná flexibilita v rozmístění, updating a podpora možností, pokročilá infrastruktura, přístroje a rysy app managementu. Společnosti, které požadují ovladatelnost, jednotnost a předpověditelnost si vybírají Windows 10 IoT Enterprise a Windows 10 Enterprise.

Windows 10 IoT Mobile Enterprise přináší schopnosti Windows 10 Mobile Enterprise do tzv. „lone-of business“ mobilních přístrojů, které spojuje přirozenou uživatelskou zkušenost se zabezpečením podnikového stupně a dobrou ovladatelností. Okamžitý přístup k aplikaci, automatická podpora skenování čárového kódu a další periferní zařízení stejně jako pocit bezpečnosti přístroje zvyšují produktivitu pro mnoho mobilních scénářů. Mobilní edice Enterprise také nabízí další schopnosti jako např. profil „multiple user“ a pokročilé uzamčení, tak aby bylo možné užívat řadu různých mobilních scénářů od maloobchodu, přes zdravotní péči, výrobu až k dalším vertikálním odvětvím.

Windows 10 IoT Core je verze Windows 10, která je optimalizována pro menší a nízkonákladová průmyslová zařízení. Jelikož jsou vytvořena tak, aby řídila přístroje jako IoT brány nebo informační stánky, je určena k řízení jedné obchodní aplikace. Windows 10 IoT Core řídí aplikaci Universal Windows a využívá stejný vývoj, konfiguraci a stejné řídicí nástroje jako

ostatní edice Windows 10 a tím usnadňuje integraci do IoT scénářů a maximální využití stávajících zdrojů.“ *Windows 10 for the Internet of Your Things* [online]. [cit. 2015-11-13]. Dostupné z: <https://www.microsoft.com/enus/WindowsForBusiness/windows-iot> (přeloženo)

2.2. Požadavky Windows 10 IoT Core

Systém je instalován obdobně jako většina jeho Linuxových konkurentů ze stolního počítače, což sebou přináší rozdílné požadavky pro běh a pro instalaci systému. Pro běh systému ve verzi RTM je požadován jednodeskový počítač Raspberry Pi 2 a Pi 3, nebo MinnowBoard MAX, DragonBoard 410c ve verzi Insider Preview je krom již zmiňovaných počítačů podporované i Intel Joule (stav k 28.10.2016), či Pi 3B+ k březnu 2018. Jako systémový disk v tomto případě slouží microSD karta, která musí být minimálně class 10 (či U1) s minimální kapacitou 8 GB, na mnoho aplikací je však doporučena karta o velikosti 16 GB. Jednodeskový počítač je potřeba připojit k domácí síti pomocí Ethernetu, nebo pomocí Wi-Fi. Pokud se rozhodneme využít Wi-Fi, tak na Raspberry Pi je potřeba oficiální USB modul (platí pro obrazy starší, než 10.0.10558, tento a novější už mají širší podporu). Na rozdíl od Wi-Fi modulů je podpora Bluetooth modulů mnohem lepší, zde neplatí žádná omezení. Pokud jde o zobrazovací jednotky, tak i v případě Raspberry doporučuji užít jednotku připojenou pomocí HDMI portu (od verze 10.0.10558 je již možné použít originální display Raspberry). Z vlastní zkušenosti mohu říct, že Display port na starších buildech funguje jen se svou vlastní ovládací aplikací, která pak umožňuje zprovoznit i dotykové ovládání, ale za cenu toho, že pokud chci spustit nějakou uživatelskou aplikaci, tak je nutné ji integrovat do aplikace pro ovládání displeje, to samé platí i pro zobrazovací jednotky připojené pomocí GPIO sběrnice.

Pokud jde o počítač, který bude provádět instalaci systému, jsou na něj kladeny jiné nároky, nežli jsou potřebné pro běh systému. Instalující počítač musí mít nainstalované Windows 10 (od verze Windows 10 IoT 10.0.10556 je možné instalaci provést ze Systému Windows 8.1 a Windows server 2012 R2) v jakékoli edici a je potřeba čtečka paměťových karet pro formát MicroSD, tedy buď USB čtečka, nebo integrovaná čtečka SD a adaptér. Poslední buildy z roku 2018 je možné nasadit i prostřednictvím Windows 11. Pro ty, kteří potřebují provést upgrade ze starší verze systému, připomínám v následující tabulce požadavky systému Windows 10. Co není nikde uvedeno, ale osobně mohu doporučit, je mít na počítači instalovaný 7zip, Winrar, či jiný nástroj pro práci s archivy. Tento nástroj se bude hodit především pro přechod na nový obraz systému Windows 10 IoT Core.

Položka	64 bitová edice		32 bitová edice	
	Minimální	Doporučené	Minimální	Doporučené
Volné místo na HDD	20 GB	30 GB	16 GB	20 GB
Velikost RAM	2 GB	4 GB	1 GB	2 GB
Procesor	1 GHz kompatibilní s Windows 8			
Síťová karta	Wi-Fi karta, nebo standartní 100 Mb, či rychlejší Ethernet			
Grafická karta	Podpora DirectX 9			

Tabulka 1: HW požadavky Windows 10 pro stolní počítač

Požadavky na PC, které bude vytvářet a překládat aplikace určené pro Windows 10 IoT Core jsou bohužel vyšší nežli minimální požadavky pro instalaci. Toto navýšení vyplývá z toho, že aplikace pro Windows 10 IoT Core jsou shodné jako pro všechny ostatní systémy rodiny Windows 10, a je potřeba použít Visual Studio 2015. Pro lepší přehlednost uvádím požadavky na systém a hardware v následující tabulce.

Položka	Minimální požadavek
Volné místo na HDD	10 GB
RAM	1 GB
Procesor	1,6 GHz
Grafická karta	DirectX 9 s rozlišením 1024 × 768
Operační systém	Windows 8.1, Windows 10, Windows Server 2012R2
Pro emulátory	64 bitový systém s Hyper-V

Tabulka 2: Požadavky Visual Studio 2015 Professional

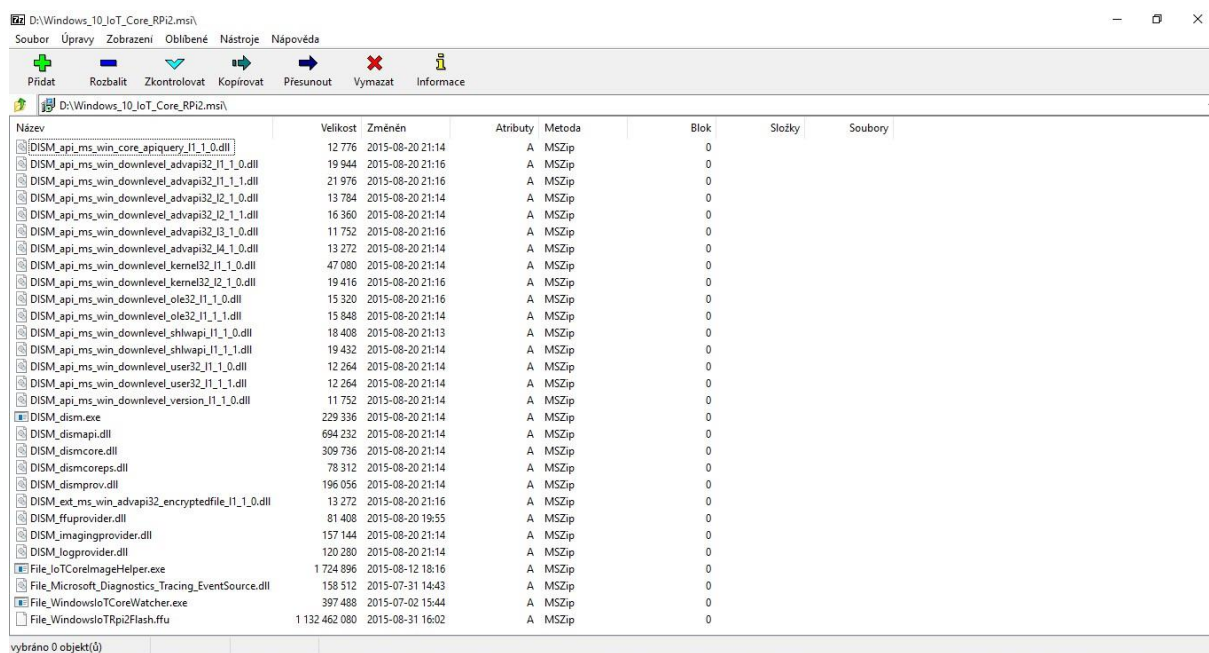
2.3. Získání systému

Na rozdíl od většiny systémů rodiny Windows je tento systém stahnutelný zcela zdarma a není pro jeho stažení potřeba ani registrace, nebo účet Microsoft. Stáhnout lze RTM i Insider Preview z Microsoft Download centra.

Pro počítače Raspberry Pi 2 a MinnowBoard MAX si můžeme vybrat dle požadované aplikace systému, kterou verzi si stáhneme, pro počítač Intel Joule máme k dispozici na konci října 2016 pouze verzi Insider Preview.

Stahování ISO souboru s potřebnými instalačními soubory nastane plně automaticky po kliknutí na požadovanou verzi systému. Bez ohledu na zvolenou verzi systému se nám vždy stáhne ISO soubor, který má na první pohled stejný obsah. Od roku 2016 je doporučeno stáhnout z docs.microsoft.com pouze aplikaci IoT Dashboard, která se postará o celý instalační proces systému včetně jeho stažení.

Uvnitř ISO souboru najdeme instalační soubor např.: Windows_IoT_Core_Rpi2.msi. Pokud si instalační soubor rozbalíme v archivačním programu, dostaneme jeho obsah, který je následující:



Název	Velikost	Změněn	Atributy	Metoda	Blok	Složky	Soubory
DISM_api_ms_win_core_apiquery_i1_1_0.dll	12 776	2015-08-20 21:14	A	MSZip	0		
DISM_api_ms_win_downlevel_advapi32_i1_1_0.dll	19 944	2015-08-20 21:16	A	MSZip	0		
DISM_api_ms_win_downlevel_advapi32_i1_1_0.dll	21 976	2015-08-20 21:16	A	MSZip	0		
DISM_api_ms_win_downlevel_advapi32_i2_1_0.dll	13 784	2015-08-20 21:14	A	MSZip	0		
DISM_api_ms_win_downlevel_advapi32_i2_1_0.dll	16 360	2015-08-20 21:14	A	MSZip	0		
DISM_api_ms_win_downlevel_advapi32_i3_1_0.dll	11 752	2015-08-20 21:16	A	MSZip	0		
DISM_api_ms_win_downlevel_advapi32_i4_1_0.dll	13 272	2015-08-20 21:14	A	MSZip	0		
DISM_api_ms_win_downlevel_kernel32_i1_1_0.dll	47 080	2015-08-20 21:14	A	MSZip	0		
DISM_api_ms_win_downlevel_kernel32_i2_1_0.dll	19 416	2015-08-20 21:16	A	MSZip	0		
DISM_api_ms_win_downlevel_ole32_i1_1_0.dll	15 320	2015-08-20 21:16	A	MSZip	0		
DISM_api_ms_win_downlevel_ole32_i1_1_0.dll	15 848	2015-08-20 21:14	A	MSZip	0		
DISM_api_ms_win_downlevel_shlwapi_i1_1_0.dll	18 408	2015-08-20 21:13	A	MSZip	0		
DISM_api_ms_win_downlevel_shlwapi_i1_1_0.dll	19 432	2015-08-20 21:14	A	MSZip	0		
DISM_api_ms_win_downlevel_user32_i1_1_0.dll	12 264	2015-08-20 21:14	A	MSZip	0		
DISM_api_ms_win_downlevel_user32_i1_1_0.dll	12 264	2015-08-20 21:14	A	MSZip	0		
DISM_api_ms_win_downlevel_version_i1_1_0.dll	11 752	2015-08-20 21:14	A	MSZip	0		
DISM_dism.exe	229 336	2015-08-20 21:14	A	MSZip	0		
DISM_dismapi.dll	694 232	2015-08-20 21:14	A	MSZip	0		
DISM_dismcore.dll	309 736	2015-08-20 21:14	A	MSZip	0		
DISM_dismcoreps.dll	78 312	2015-08-20 21:14	A	MSZip	0		
DISM_dismprov.dll	196 056	2015-08-20 21:14	A	MSZip	0		
DISM_ext_ms_win_advapi32_encryptedfile_i1_1_0.dll	13 272	2015-08-20 21:16	A	MSZip	0		
DISM_ffupprovider.dll	81 408	2015-08-20 19:55	A	MSZip	0		
DISM_imagingprovider.dll	157 144	2015-08-20 21:14	A	MSZip	0		
DISM_logprovider.dll	120 280	2015-08-20 21:14	A	MSZip	0		
File_IoTCoreImageHelper.exe	1 724 896	2015-08-12 18:16	A	MSZip	0		
File_Microsoft_Diagnostics_Tracing_EventSource.dll	158 512	2015-07-31 14:43	A	MSZip	0		
File_WindowsIoTCoreWatcher.exe	397 488	2015-07-02 15:44	A	MSZip	0		
File_WindowsIoTRpi2Flash.ffu	1 132 462 080	2015-08-31 16:02	A	MSZip	0		

Obr. 1: Obsah instalátoru Windows 10 IoT Core

Z obsahu tohoto souboru nás zajímá nejvíce soubor File_WindowsIoTRpi2.ffu, který je obdobou souboru Install.wim, jež se nachází na instalačním médiu Windows ve složce Sources. Formát ffu, tedy Full Flash Update, značí v podání Microsoftu, že se jedná o instalační soubor zahrnující také příslušný firmware a ovladače pro dané zařízení. Když si pomocí archivačního

programu rozbalíme soubor File_WindowsIoTRpi2.ffu, získáme stejnou strukturu, jako u souboru Install.wim a jako se nám vytvoří na paměťové kartě při instalaci systému. Jak je možné si všimnout, složky Program Files(x86) a Program Files jsou oddělené, i když systém pracuje na architektuře ARM, protože existují i verze pro x86 a x64.

Název	Velikost	Komprimovan...	Změněn	Vytvořen	Použit	Atributy	Krátké jméno	Složky	Soubory
boot	3 170 304	3 170 304	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	D	boot	0	1
EFI	2 562 399	2 564 096	2015-08-31 15:58	2015-08-31 15:58	2015-08-31 00:00	D	EFI	3	5
Program Files	0	0	2015-08-31 15:58	2015-08-31 15:58	2015-08-31 00:00	D	PROGRA~1	1	0
Program Files (x86)	0	0	2015-08-31 15:58	2015-08-31 15:58	2015-08-31 00:00	D	PROGRA~2	1	0
Users	8 192	8 192	2015-08-31 15:58	2015-08-31 15:58	2015-08-31 00:00	D	USERS	1	3
Windows	959 770	997 376	2015-08-31 15:58	2015-08-31 15:58	2015-08-31 00:00	D	WINDOWS	25	72
bootcode.bin	17 900	18 432	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	bootcode.bin		
config.txt	393	1 024	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	config.txt		
fixup.dat	6 159	7 168	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	fixup.dat		
fixup_cd.dat	2 362	3 072	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	fixup_cd.dat		
fixup_x.dat	9 213	9 216	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	fixup_x.dat		
kernel.img	884 736	884 736	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	kernel.img		
LICENCE.broadcom	1 476	2 048	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	LICENC~1.BRO		
start.elf	2 656 696	2 657 280	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	start.elf		
start_cd.elf	560 344	561 152	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	start_cd.elf		
start_x.elf	3 614 440	3 614 720	2015-08-31 15:59	2015-08-31 15:59	2015-08-31 00:00	A	start_x.elf		

Obr. 2: Obsah souboru File_WindowsIoTRpi2.ffu

Je nutné podotknout, že celý instalátor má velmi dobrou kompresi vnitřních souborů, jen soubor File_WindowsIoTRpi2.ffu má vysoký kompresní poměr a jeho plná velikost je 1132,5 MB, přitom velikost celého instalátoru, tedy i se všemi programy, které se instalují do PC, a podpůrnými knihovnami, má jen 511,43 MB. Při bližším zkoumání nalezneme i další analogie s plnou verzí systému Windows, třeba aplikace DISM.exe, jejíž grafickou nadstavbou je IoT Dashboadr. Aplikace DISM.exe slouží k práci s obrazy systému Windows a mimo jiné jej umožňuje aplikovat na zvolený disk. V případě IoT WindowsIoTRpi2.ffu a v případě počítačové verze Install.wim na cílové médium, které může být připojeno pomocí USB, tedy nemusí se jednat, a v případě Windows IoT se ani nejedná, o systémový disk.

K plnohodnotné práci se systémem již potřebujeme jen Visual Studio, které si můžeme zakoupit ve verzi Professional nebo Enterprise (pouze s MSDN), či si stáhnout bezplatnou verzi. V případě bezplatné verze Visual Studio Community 2015, které je možné doplnit i o český jazykový balíček je pro aktivaci vyžadován e-mailový účet společnosti Microsoft. Visual Studio Community je dostupné zdarma pro samostatné vývojáře, vývoj Open Source, akademický výzkum, vzdělávací instituce a malé profesionální týmy (do 5 programátorů včetně).

Pro práci s Windows 10 on ARM v rámci projektu WoR je potřeba pro nasazení a úpravy projektů Visual Studio 2019, nebo 2022, starší verze nemají ARM64 debugovací nástroje.

2.3.1. Užitý systém pro práci

Pro vypracování teoretické části bakalářské práce jsem si zvolil Insider Preview ze dne 14.10.2015. Jedná se o verzi systému 10.0.10531. Pro vypracování praktické části a srovnání rychlosti vývoje jsem použil verzi systému 10.0.14342.1000 ze dne 8.5.2016.

Jako HW základní mám k dispozici microSDHC kartu rychlosti U1 a kapacitě 16 GB od Verbatimu, jednodeskový počítač je Raspberry Pi2, Wi-Fi komunikaci obstarává originální Raspberry USB Wi-Fi Dongle a zobrazovací jednotkou je 7“ displej připojený pomocí HDMI. Aplikace jsou vyvíjeny ve Visual Studiu 2015 Enterprise běžícím na Windows 10 Enterprise x64, či Windows Server 2016 Datacenter TP5, s aktivním Hyper-V.

2.4. Specifika Windows 10 IoT Core

Proti svým Linuxovým konkurentům má tento systém dvě velká specifika, kterých si všimne každý uživatel. Prvním specifikem je, že systém umožňuje mít spuštěnou jen jednu aplikaci typu Windows Universal (desktopové aplikace nejsou podporovány). Tato vlastnost má v první řadě zajistit stálý výkon HW pro řízení výstupních jednotek. Další na první pohled viditelným rozdílem je to, že vlastní ovládání systému neprobíhá grafickým rozhraním, ale vzdáleně z jiného počítače pomocí webového serveru umístěného v systému. Z výchozí aplikace systému (jak u RTM, tak u Insider Preview) je potřeba nastavit jazyk systému a je možné konfigurovat připojení k Wi-Fi, dále lze otevírat manuály a zobrazit si adresy nápovědy. Výchozí aplikace především vypisuje důležité informace o systému, jako je jméno počítače, IP adresa verze 4 a 6, MAC adresa a verze systému.

Co je u Windows 10 IoT Core pravidlem a u Linuxové konkurence možností, je bootování do konkrétní aplikace. V případě Windows lze aplikaci přepnout pomocí webserveru nebo zapnutím ladícího programu Visual Studio. V případě přerušení aplikace ladícím programem Visual Studio se po jeho ukončení systém vrací k původní aplikaci. V případě kolapsu systémového procesu nebo chybě aplikace (nemusí to být nutně úplný pád, stačí stav, který je na desktopových aplikacích označován jako „neodpovídá“) dojde k automatickému restartování systému (není pravidlem pro kolaps aplikace nasazené ladícím programem Visual Studio), zpravidla po 50 vteřinách (pokud nenastavíme jinak a systém problém nevyřeší) a k opětovnému spuštění výchozí aplikace, i v případě, že zkolabovala aplikace ručně spuštěná pomocí webového rozhraní. Pokud jde o konfiguraci pomocí SSH u Linuxových konkurentů, má

Windows obdobu v konfiguraci pomocí PowerShellu 5, jde to i pomocí starší verze, ale tato verze umožňuje tvorbu a sdílení modulů pro práci se systémem. Pokud chci, aby systém prováděl výhradně operace zadávané pomocí PowerShellu, existuje mezi vzorovými projekty od Microsoftu též projekt umožňující toto nastavení systému, ale pro systémovou konfiguraci, jak si ukážeme dále, není potřeba nic víc, nežli znát IP adresu nebo jméno počítače, a můžeme psát příkazy v PowerShellu.

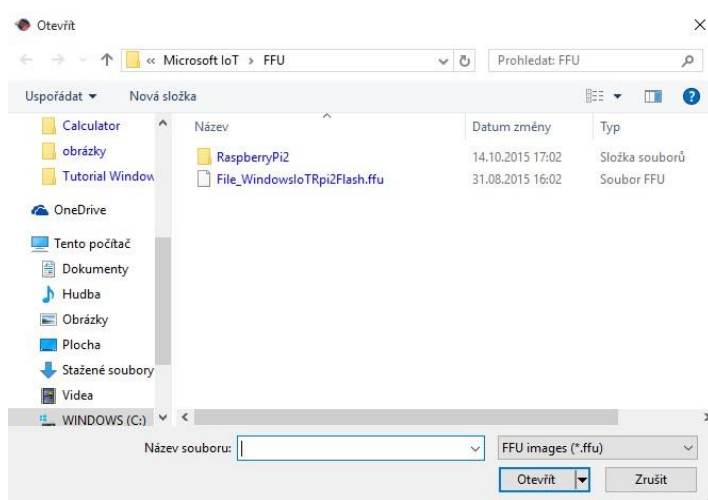
K systému je možné se připojit i pomocí SSH, ale je nutné si uvědomit, že se připojujeme k PowerShellu, nikoliv k Bashi.

Tím nejvýraznějším rozdílem proti Linuxové konkurenci je chráněné jádro systému založené na technologii NT, které je pro tento systém obdobné jako v případě mobilní verze. Systém jako takový podporuje Plug and Play pro většinu externího hardwaru připojeného pomocí USB. Rozšiřující knihovny umožňují práci s GPIO sběrnici.

Na rozdíl od zbytku rodiny systémů Windows 10 není tento systém vybaven přístupem do Microsoft Store a možnosti užít účet Microsoft jako uživatelský účet.

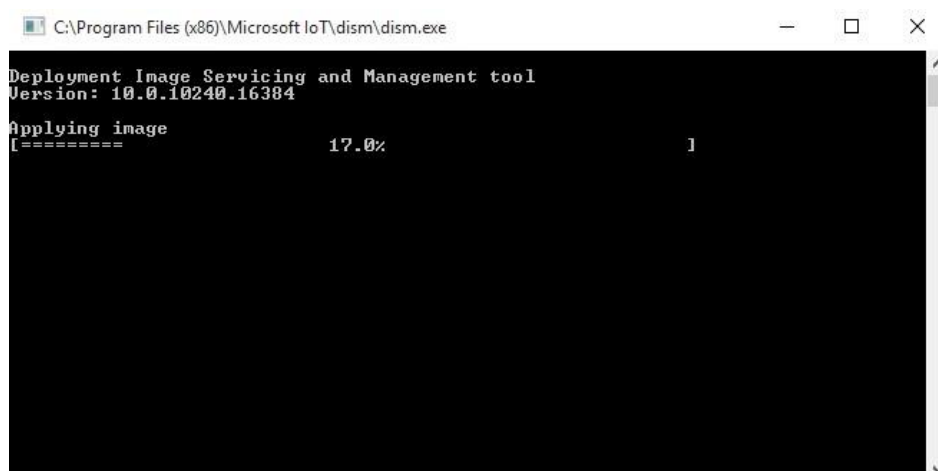
2.5. Instalace systému Windows 10 IoT Core

Instalace systému na paměťovou kartu je začátkem práce se systémem. Nyní stačí jen připojit připravenou microSD kartu do čtečky karet v počítači, nebo pomocí USB portu. Pokud máme na kartě, kterou chceme použít, nějaké soubory, přkopírujeme si je do počítače, abychom o ně nepřišli. Pokud vybíráme obraz systému, který máme rozbalený z MSI instalátoru, nezapomenout, že má formát ffu.



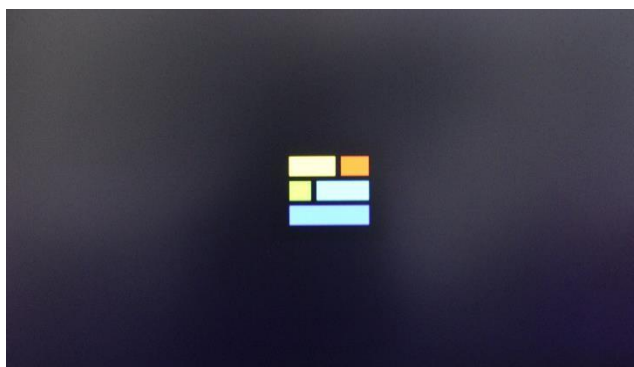
Obr. 3: Vybrání souboru obrazu systému, který bude instalován

Pokud spustíme vlastní instalaci, dojde k zavolání programu `dism.exe`, který je nativní součástí systému Windows. Tento program běží v příkazové řádce a provede vlastní vytvoření systémové jednotky ze zvolené karty.



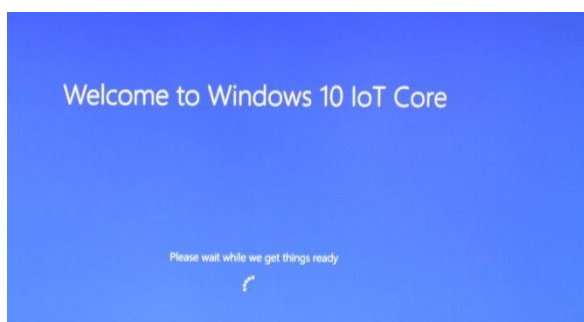
Obr. 4: Program `dism.exe`

Po vložení paměťové karty připojíme k jednodeskovému počítači všechny potřebné periferie, i když pro začátek doporučuji omezit se na Ethernet či Wi-Fi, myš, klávesnici a eventuálně Bluetooth, když bychom si chtěli spárovat periferie. Co budeme potřebovat zcela jistě, je připojit zobrazovací jednotku, monitor, televizi, nebo displej pomocí HDMI. Pokud máme všechno připraveno, můžeme připojit napájení a spustit počítač. Při spouštění uvidíme na obrazovce logo Windows stejně jako v případě, když spouštíme Windows 8 a novější verze na klasickém PC. Následuje načítání výchozí aplikace (viz obrázek 5). Když je na obrazovce toto logo, je systém již načtený, ale spouští zvolenou výchozí aplikaci.



Obr. 5: Načítání aplikace na Windows 10 IoT Core

Jako první uvidíme pomocnou konfigurační aplikaci, období zdlouhavého dokončování instalace Windows do klasického počítače, ze kterého se systém dostane na plochu. Tato pomocná aplikace je ukázána na obrázku 6.



Obr. 6: Pomocná aplikace konfigurace OS

Aplikace nám poslouží pro nastavení jazyka zobrazení systému a nastavení Wi-Fi připojení. Po dokončení této konfigurace systém přejde do standardní výchozí aplikace, kterou uvidíme již při každém dalším spuštění, pokud nenastavíme jinak.

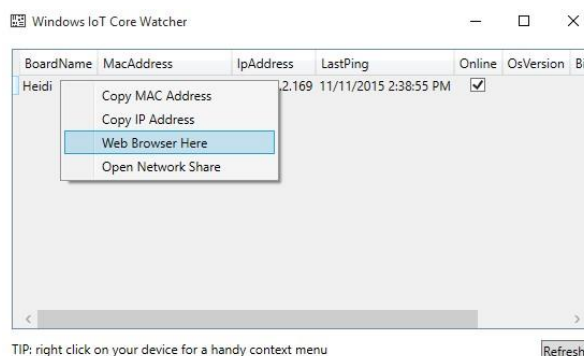
Upgrade na nový obraz systému se provede velmi jednoduše, po stažení souborů z paměťové karty provede instalaci novější verze systému a následně vrátíme potřebné soubory zpět. Po dokončení instalace se nám zobrazí výchozí aplikace systému (viz obr. 7).



Obr. 7: Výchozí aplikace, pokud je systém připojen k Wi-Fi i Ethernetu

2.5.1. Základní konfigurace Windows 10 IoT Core

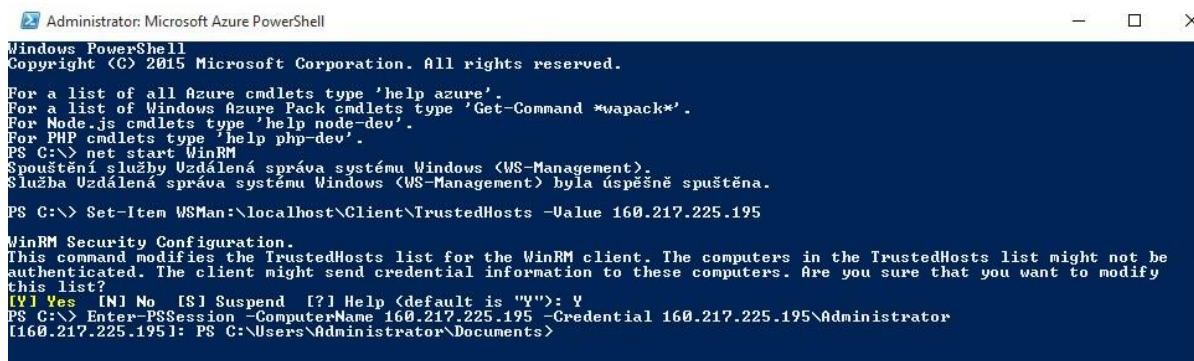
Pro vyhledání zařízení v síti máme dvě možnosti, buď opsat ze zobrazovače zobrazenou IP adresu, nebo si spustit IoTCoreWatcher. Po nalezení zařízení pomocí IoTCoreWatcheru stačí kliknout pravým tlačítkem na název počítače a zvolit kopírovat IP adresu (viz obrázek 8). Od roku 2016 pak využíváme položku My Devices v aplikaci IoT Dashboard, nebo aplikaci Windows IoT Remote Client.



Obr. 8: WindowsIoTCoreWatcher

Po instalaci má systém nastavené výchozí hodnoty pro přihlášení. Uživatelský účet je *Administrator* a heslo k němu je: p@ssw0rd

Pokud byl systém instalován pomocí aplikace IoT Dashboard, nese výchozí uživatelský účet Administrator námi definované heslo. Výchozí heslo je platné pro instalaci pomocí aplikace Image Helper, nebo příkazové řádky. Pro změnu výchozího hesla je možné použít PowerShell, který je potřeba mít spuštěný jako správce, pomocí vzdáleného připojení k Windows 10 IoT Core. Na následujícím obrázku je vidět postup pro připojení se k Windows 10 IoT Core pomocí PowerShellu. Doporučuji užít základní systémový PowerShell, kdo užije (jak je na obrázku ukázáno) Azure PowerShell, nic se neděje, jen nemá tak dobře barevně vyznačenou syntaxi.



Obr. 9: Přihlášení pomocí PowerShellu k Windows 10 IoT Core

Pokud jsme přihlášení, užijeme příkaz: „net user Administrator [new password]“ pro provedení změny hesla účtu administrátora. Na obrázku 10 nalezneme příklad použití tohoto příkazu, jak je aplikován při vypracování méj bakalářské práce.

```
[160.217.225.195]: PS C:\Users\Administrator\Documents> net user Administrator [LM27C256]
The command completed successfully.
[160.217.225.195]: PS C:\Users\Administrator\Documents>
```

Obr. 10: Změna hesla administrátora z p@ssw0rd na LM27C256

Pro změnu názvu zařízení z výchozího minwinpc na náš libovolný (v mém případě na Heidi) použijeme příkaz: „setcomputername Heidi“. Po provedení tohoto příkazu je nutné zařízení restartovat či vypnout. Vypnutí zařízení je ukázáno na obrázku 11, pokud budeme chtít počítač restartovat, příkaz budeme modifikovat tak, že místo „/s“ napíšeme „/t“. Parametr „/t 0“ značí, že restart proběhne hned, pokud si přejeme jiný čas, tak místo 0 dosadíme jiný čas v sekundách, kdy největší možné oddálení je 10 let, tedy 315360000 sekund. Velkým problémem je, že se špatně poznává okamžik, kdy je systém skutečně vypnutý, protože kontrolky na zařízení svítí hodně dlouho a PowerShell neumožní spustit po tomto příkazu v rámci relace nic dalšího. Z vlastní zkušenosti mohu říct, že, když systém přejde do stavu off-line dle IoTCoreWatcheru, je prakticky vypnutý. V další podkapitole uvedu, jak toto poznat při práci ve webovém rozhraní.

```
[160.217.225.195]: PS C:\Users\Administrator\Documents> shutdown /s /t 0
System will shutdown in 0 seconds...
[160.217.225.195]: PS C:\Users\Administrator\Documents>
```

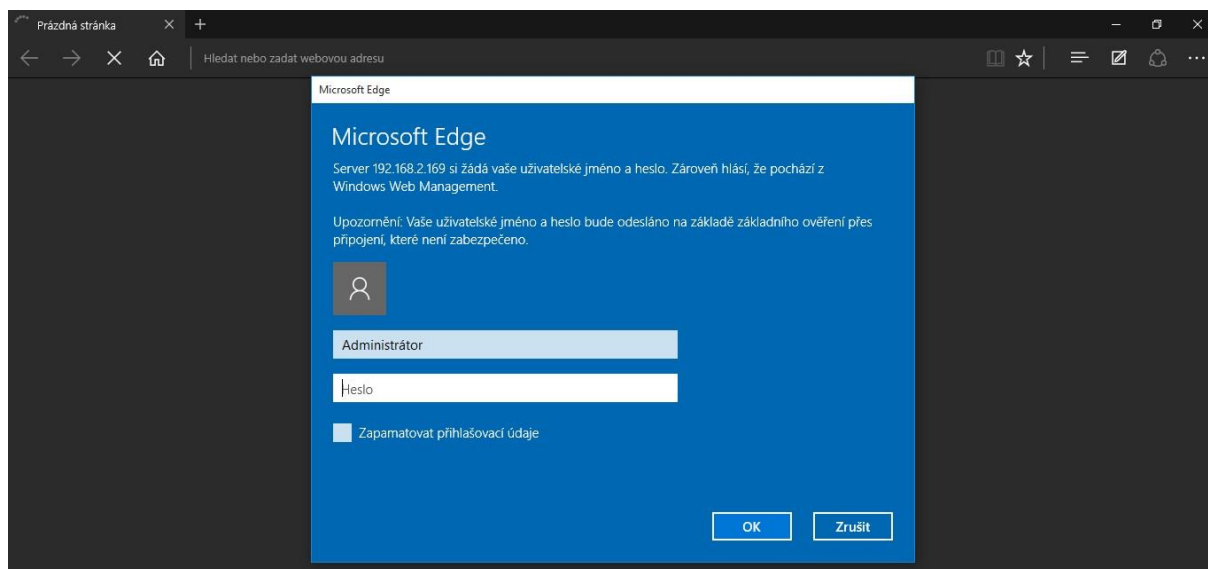
Obr. 11: Vypnutí počítače Raspberry pomocí PowerShellu

V PowerShellu opustíme relaci pomocí příkazu „exit“ a pro připojení se k zařízení po restartu musíme začít od příkazu: „Set-Item WSMAN:\localhost\Client\TrustedHosts -Value Heidi“, (uvedený tvar je konkrétní příklad) pokud jsme místo IP adresy použili název počítače, pokud jsme použili IP adresu, je možné rovnou vyvolat vzdálené připojení.

2.5.2. Webové rozhraní pro řízení systému

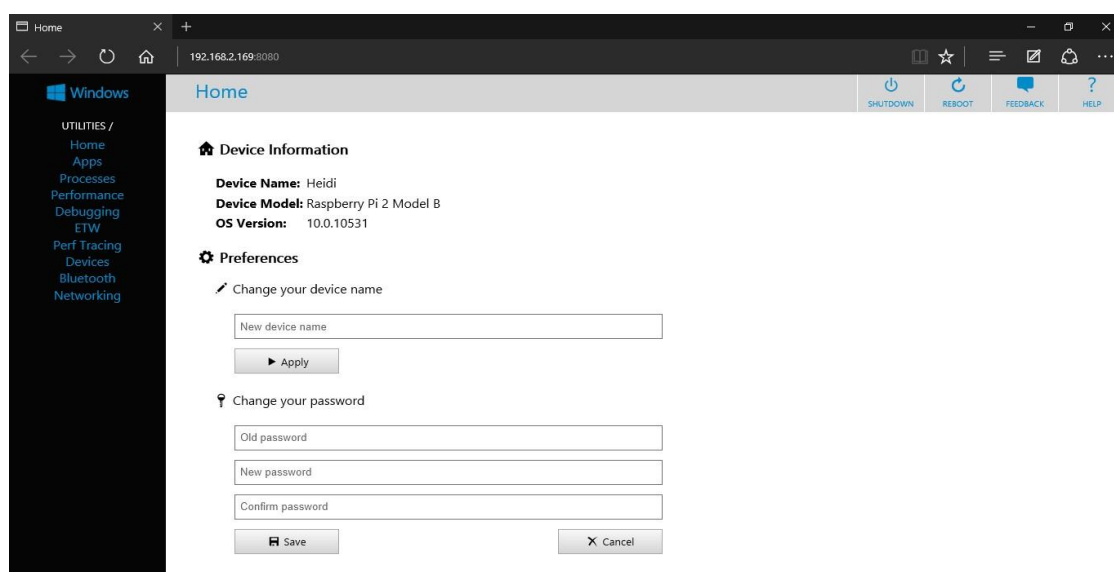
Mnohem pohodlnější možností, jak pracovat s Windows 10 IoT Core je použít integrovaného webového rozhraní pro ovládání systému. Jsou teoreticky dvě možnosti, tou první je opsání IP adresy (myslím verzi 4) do adresního řádku webového prohlížeče, druhou je užít oblíbený IoTCoreWatcher a kliknout na jméno počítače pravým tlačítkem a zvolit „Web Browse Here“. Mně se prakticky osvědčila jen druhá zmiňovaná cesta, u té první se mi často stává, že skončím hlášením „Stránku nelze zobrazit“. Nicméně obě cesty vedou na přihlášení se

k systému pomocí webserveru, jak je vidět na obrázku 12. Do formuláře zadáváme jak uživatelské jméno, tak heslo, protože si můžeme vytvořit i další uživatele.



Obr. 12: Přihlášení k Windows 10 IoT Core pomocí Microsoft Edge

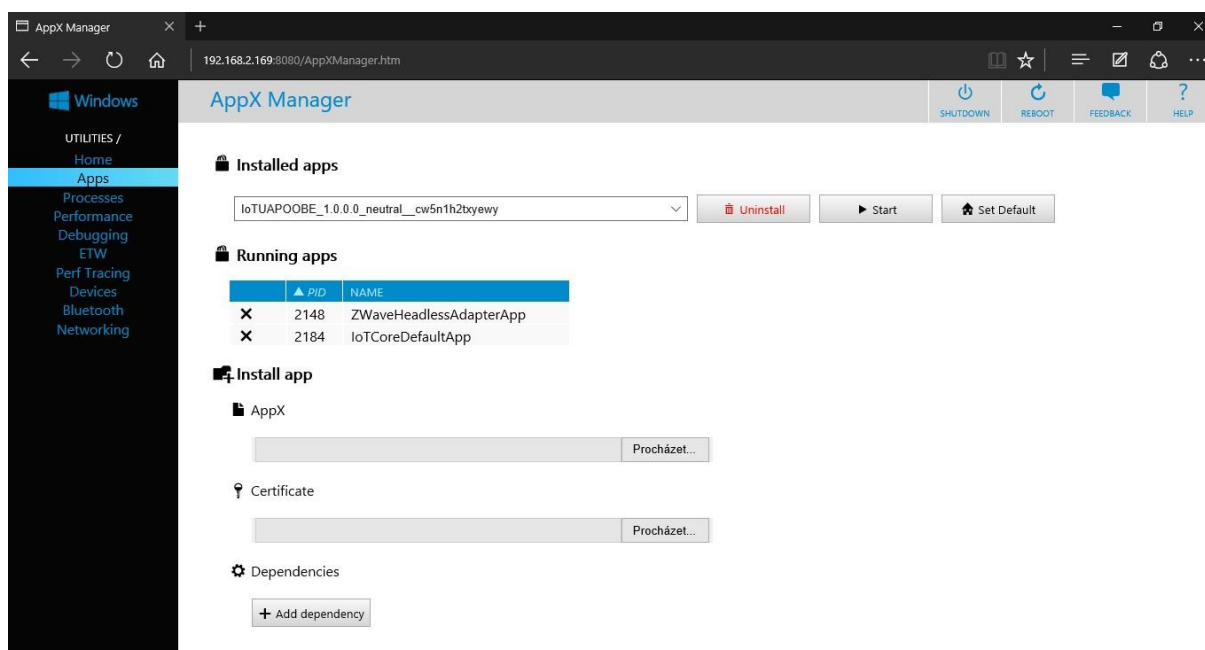
Po přihlášení se nám zobrazí stránka Utilities, která nám umožňuje provést všechna nastavení, která jsme dělali v minulé podkapitole pomocí PowerShellu, tedy umožňuje změnit jméno zařízení a změnit heslo přihlášeného uživatele. Ve skutečnosti však nejde o stránku Utilities, ale o stránku Home, protože Utilities není stránkou, ale adresářem. V horním neměnném pásu je možnost rozbalit si nápovědu, poslat zpětnou vazbu do Microsoftu a restartovat, či vypnout zařízení. Když jsem se opět dostal k problematice vypínání, je dobré vypínat, když jsme připojeni pomocí webového rozhraní na stránce Performance, protože se zmizením křivek z grafů můžeme zařízení považovat za vypnuté.



Obr. 13: Úvodní stránka webového rozhraní

3.5.2.1. Aplikace

Stránka Apps nám umožní práci s univerzálními aplikacemi pro Windows. Na stránce je možné nahrát balíček aplikace, který si vytvoříme pomocí Visual Studia, zvolit výchozí aplikaci, která se spustí se startem systému a také zvolit aplikaci, která poběží v daný okamžik, tedy spustit konkrétní aplikaci ze seznamu nainstalovaných aplikací. Na této stránce je možné aplikace i odinstalovat či zastavit.

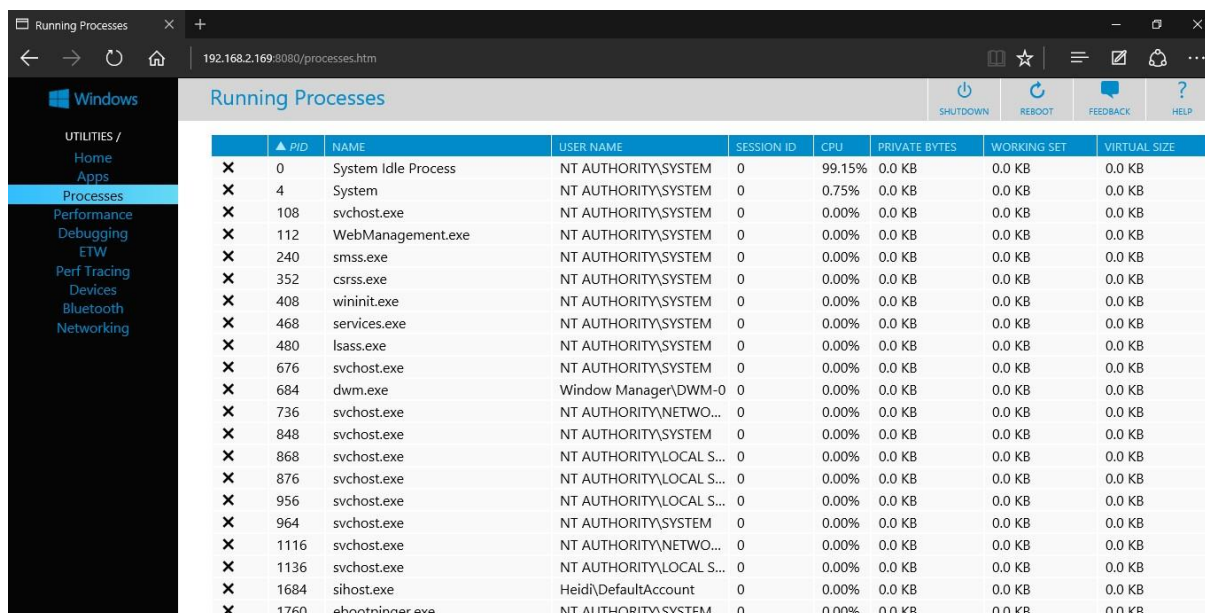


Obr. 14: Stránka Apps

Pokud máme vytvořený balíček aplikace a certifikát k aplikaci, přihlásíme se do webového rozhraní systému a přejdeme na záložku „Apps“ a ve spodních 2/3 stránky je sekce, která slouží k instalaci nových aplikací do systému. Do první kolony vložíme cestu k balíčku s příponou appx stejným způsobem, jako když bychom chtěli nahrát soubor na kteroukoli jinou webovou stránku. Do druhého slotu vložíme obdobně cestu k certifikátu a všechno stvrdíme úplně dole tlačítkem „Go“. Nyní stačí počkat, až se balíčky přenesou do zařízení, což trvá o něco déle nežli tomu při vzdáleném ladění pomocí Visual Studia. Po dokončení nahrávání (obnoví se stránka) si aplikaci vyhledáme v horní části stránky, konkrétně v sektoru Installed Apps, v seznamu, který si zobrazíme rozkliknutím. Následně doporučuji nejdříve systém přepnout do naší aplikace pomocí tlačítka „start“ a po ověření, že aplikace pracuje, jak má, ji teprve nastavit jako výchozí tlačítkem „Set Default“.

3.5.2.2. Procesy

Velmi důležitou je stránka Processes, která zobrazuje podrobný výpis běžících procesů, umožňuje jejich ukončování a ve spodní části má i okno umožňující spouštění procesů či příkazů. Proces či příkaz se spouští s právy přihlášeného uživatele.

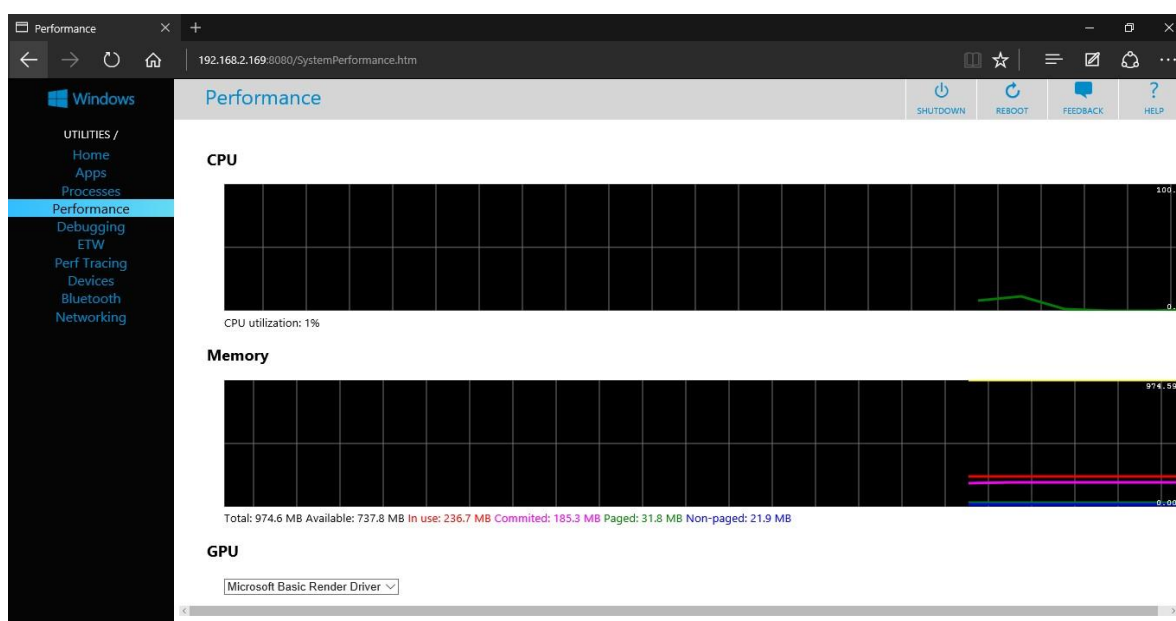


PID	NAME	USER NAME	SESSION ID	CPU	PRIVATE BYTES	WORKING SET	VIRTUAL SIZE
0	System Idle Process	NT AUTHORITY\SYSTEM	0	99.15%	0.0 KB	0.0 KB	0.0 KB
4	System	NT AUTHORITY\SYSTEM	0	0.75%	0.0 KB	0.0 KB	0.0 KB
108	svchost.exe	NT AUTHORITY\SYSTEM	0	0.00%	0.0 KB	0.0 KB	0.0 KB
112	WebManagement.exe	NT AUTHORITY\SYSTEM	0	0.00%	0.0 KB	0.0 KB	0.0 KB
240	smss.exe	NT AUTHORITY\SYSTEM	0	0.00%	0.0 KB	0.0 KB	0.0 KB
352	csrss.exe	NT AUTHORITY\SYSTEM	0	0.00%	0.0 KB	0.0 KB	0.0 KB
408	wininit.exe	NT AUTHORITY\SYSTEM	0	0.00%	0.0 KB	0.0 KB	0.0 KB
468	services.exe	NT AUTHORITY\SYSTEM	0	0.00%	0.0 KB	0.0 KB	0.0 KB
480	lsass.exe	NT AUTHORITY\SYSTEM	0	0.00%	0.0 KB	0.0 KB	0.0 KB
676	svchost.exe	NT AUTHORITY\SYSTEM	0	0.00%	0.0 KB	0.0 KB	0.0 KB
684	dwm.exe	Window Manager\DWM-0	0	0.00%	0.0 KB	0.0 KB	0.0 KB
736	svchost.exe	NT AUTHORITY\NETWO...	0	0.00%	0.0 KB	0.0 KB	0.0 KB
848	svchost.exe	NT AUTHORITY\SYSTEM	0	0.00%	0.0 KB	0.0 KB	0.0 KB
868	svchost.exe	NT AUTHORITY\LOCAL S...	0	0.00%	0.0 KB	0.0 KB	0.0 KB
876	svchost.exe	NT AUTHORITY\LOCAL S...	0	0.00%	0.0 KB	0.0 KB	0.0 KB
956	svchost.exe	NT AUTHORITY\LOCAL S...	0	0.00%	0.0 KB	0.0 KB	0.0 KB
964	svchost.exe	NT AUTHORITY\SYSTEM	0	0.00%	0.0 KB	0.0 KB	0.0 KB
1116	svchost.exe	NT AUTHORITY\NETWO...	0	0.00%	0.0 KB	0.0 KB	0.0 KB
1136	svchost.exe	NT AUTHORITY\LOCAL S...	0	0.00%	0.0 KB	0.0 KB	0.0 KB
1684	sihost.exe	Heidi\DefaultAccount	0	0.00%	0.0 KB	0.0 KB	0.0 KB
1760	ebootpinger.exe	NT AUTHORITY\SYSTEM	0	0.00%	0.0 KB	0.0 KB	0.0 KB

Obr. 15: Stránka procesů

3.5.2.3. Sledování výkonu

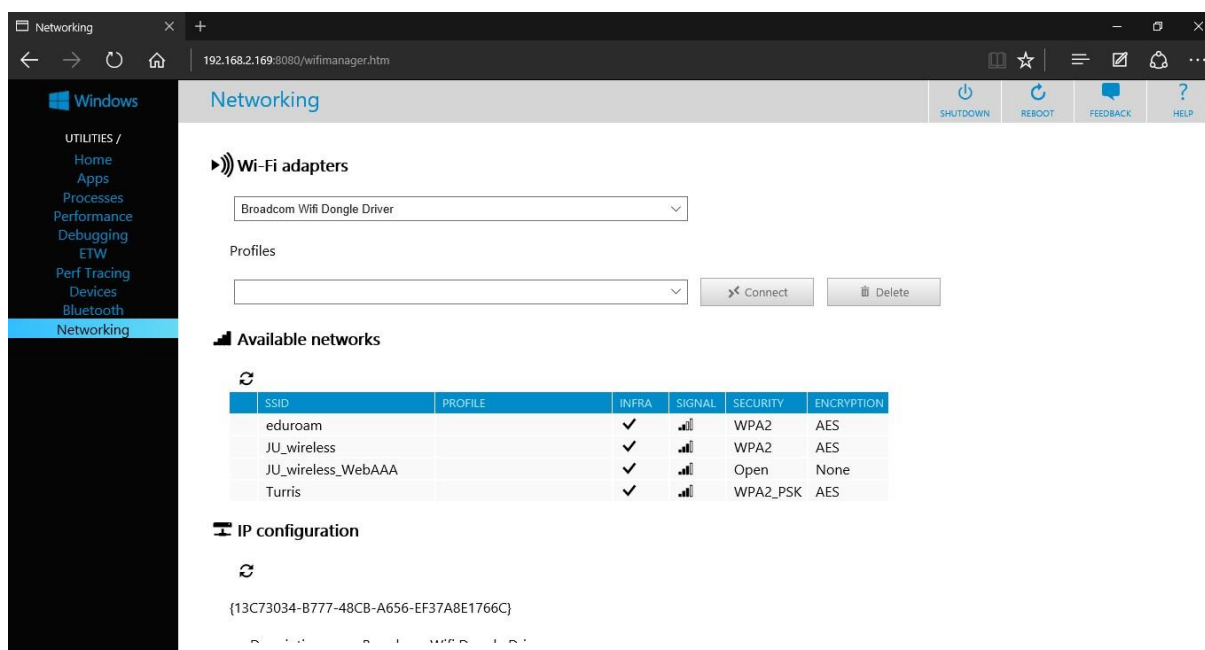
Ke sledování výkonu použijeme stránku Performance, kde je možné graficky i číselně zjistit aktuální využití systémových prostředků, externích síťových a grafických karet.



Obr. 16: Stránka Performance

3.5.2.4. Správa sítě

Poslední stránkou, která je z hlediska běžného užívání systému velmi důležitou je stránka Networking, kde je možné nastavit Wi-Fi připojení, zobrazit aktuální konfiguraci síťového připojení a v případě více zařízení pro síťový přístup mezi těmito zařízeními přepínat. Dále je možné na této stránce spravovat profily sítí. Pro informace o ovladačích a o tom, kde je, co připojeno, použijeme stránku Devices.



Obr. 17: Stránka pro konfiguraci síťového připojení

2.6. Práce se systémem pomocí PowerShellu

PowerShell je velmi mocný nástroj pro práci se systémem Windows a v případě Windows IoT to platí obzvláště. Pokud chceme v tomto systému třeba vytvořit nového uživatele, nemáme v podstatě jinou možnost nežli použít PowerShell. I při zpracování této práce jsem PowerShell hodně používal, takže nyní chci uvést pár užitečných příkazů, které se mohou hodit při práci se systémem. Všechny příkazy, které tu uvádím, předpokládají, že je uživatel již připojený pomocí vzdálené relace k systému.

2.6.1. Procesy

Prvním užitečným příkazem je „tlist“, který vypíše seznam běžících procesů a jejich ID. Jeho lepší dvojčce je příkaz „get-process“, který vypíše podrobné informace o běžících procesech, obdobně jako je tomu u webového rozhraní. Jak vypadá výstup z tohoto příkazu, je vidět na obrázku 18. Pro ukončení procesu užijeme příkaz „kill.exe [jméno či ID procesu]“.

[160.217.225.195]: PS C:\Users\Administrator\Documents> **get-process**

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
235	8	2544	11000	42	0.52	1132	backgroundTaskHost
153	5	396	1440	12	0.34	372	csrss
388	7	5768	13500	53	0.59	612	dwm
52	410	428	1180	9	0.67	1420	ebootpinger
41	4	236	928	8	0.06	1444	ftpd
0	0	0	8	0		0	Idle
512	17	7084	32448	221	2.80	1984	IoTCoreDefaultApp
599	9	1676	7072	25	2.45	500	lsass
83	3	588	1032	18	0.20	1484	msvsmom
105	4	684	992	22	0.14	2604	msvsmom
182	7	2800	10904	39	8.09	1296	RuntimeBroker
523	23	4696	13868	72	7.45	548	SearchIndexer
179	4	1296	3568	10	2.05	480	services
410	10	5480	18004	68	1.23	1312	sihost
44	1	140	736	3	0.25	248	smss
319	8	2260	8324	32	2.73	600	svchost
370	8	1284	4324	19	1.36	660	svchost
277	23	3168	8452	44	1.69	784	svchost
401	12	5600	10428	51	2.61	812	svchost
546	23	4192	10824	92	7.05	836	svchost
589	15	5296	16740	68	4.45	896	svchost
462	11	3460	12152	58	3.48	956	svchost
539	18	4760	11204	47	3.59	976	svchost
84	3	724	3324	16	0.08	1192	svchost
272	9	4516	12084	71	3.97	1584	svchost
118	3	1208	5360	28	0.34	1700	svchost
127	5	1028	5648	33	0.61	1716	svchost
95	5	744	3132	15	0.08	1748	svchost
452	0	36	64	1	18.41	4	System
107	5	1472	5096	23	0.39	1740	WebManagement
74	4	552	2744	14	0.16	432	wininit
661	33	45456	66816	131	85.81	3520	wsmprovhost

Obr. 18: Výstup příkazu Get-Process

Následující tabulka vysvětlí význam zkratk, které se objevují ve výpisu.

Název zdroje	Zkratka	Popis
Handles	žádný	Popisovač umístění v paměti. Při zavření popisovače je paměť uvolněna.
Nestránkovaný fond (v kilobajtech)	NPM(k)	Nestránkovaný fond je ukládání paměti, která není nikdy stránkována na pevný disk, takže je to rychleji dostupné.
Stránkovaný fond (v kilobajtech)	PM(k)	Stránkovaný fond může zaslat disk v případě, že místo je vyžadováno. Díky stránkovanému fondu je místo účinně větší než u nestránkovaného fondu. Některé čtení paměti může trvat déle, protože požadovaná data jsou nyní uložena na disku.
Pracovní sada (v kilobajtech)	WS(k)	Pracovní sada je sada stránek fyzické paměti, kterou proces používá. Pouze data uložená ve fyzické paměti (momentálně stránkování na disk) je v pracovní sadě.
Virtuální paměť (v megabajtech)	VM(M)	Velikost virtuální paměti přidělené pro výhradní použití v procesu.
Čas procesoru (v sekundách)	CPU (s)	Množství času procesoru, které proces používá (na všech dostupných procesorech).
ID procesu	žádný	Jedinečný identifikátor pro daný proces. I na sdílený počítač bude mít pouze jeden proces daným identifikátorem.
Název procesu	žádný	Popisný identifikátor procesu, ale na rozdíl od ID procesu nikoli nutně jedinečný.

Tab. 3: Vysvětlení zkratk ve výpisu příkazu *Get-Process*

2.6.2. Připojení k síti

Pokud chceme vidět, jak vypadá aktuální stav připojení systému k síti, zda je systém připojený pomocí Wi-Fi, nebo Ethernetu, či jaký rychlostní standard podporuje síťová karta pro Ethernet, je na místě použít příkaz „Get-NetIPConfiguration“, jak je vidět na obrázku.

```
[160.217.225.195]: PS C:\Users\Administrator\Documents> get-NetIPConfiguration

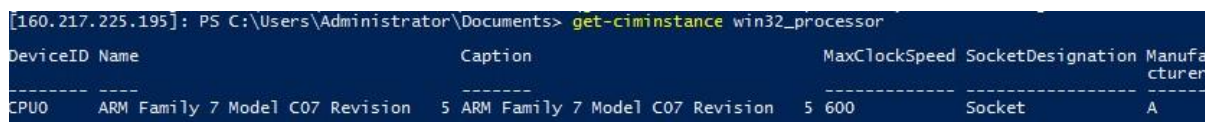
InterfaceAlias      : Ethernet
InterfaceIndex      : 5
InterfaceDescription : LAN9512/LAN9514 USB 2.0 to Ethernet 10/100 Adapter
IPv4Address          : 160.217.225.195
IPv6DefaultGateway  : 
IPv4DefaultGateway  : 160.217.225.254

InterfaceAlias      : Wi-Fi
InterfaceIndex      : 4
InterfaceDescription : Broadcom Wifi Dongle Driver
NetAdapter.Status    : Disconnected
```

Obr. 19: Výpis IP konfigurace systému

2.6.3. Výpis informací o procesoru

Pokud chceme spustit vzdálené ladění z Visual Studia pro nějakou aplikaci a nevíme, zda systém běží na ARM, nebo na x86 či x64, není lepší možnost, nežli se připojit pomocí PowerShellu ke vzdálenému počítači a dát příkaz „get-ciminstance win32_processor“, který nám vrátí architekturu procesoru, jak je vidět na obrázku 20. Tento příkaz funguje v jakékoliv verzi systému Windows, včetně mobilních systémů, Windows CE a Windows Embedded. Jen pozor, vypisuje opravdu architekturu procesoru, nikoliv OS.



```
[160.217.225.195]: PS C:\Users\Administrator\Documents> get-ciminstance win32_processor
```

DeviceID	Name	Caption	MaxClockSpeed	SocketDesignation	Manufacturer
CPU0	ARM Family 7 Model C07 Revision 5	ARM Family 7 Model C07 Revision 5	600	Socket	A

Obr. 20: Zobrazení architektury procesoru pomocí PowerShell

2.6.4. Správa uživatelských účtů

Kromě výchozího účtu administrátora si můžeme vytvořit další uživatelský účet, a to jako na běžném počítači s právy administrátora, nebo s právy běžného uživatele. Na rozdíl od plných Windows 10 jsou všechny uživatelské účty jen lokální. Pro vytvoření účtu běžného uživatele použijeme příkaz „net user [uživatelské jméno] [heslo] /add“ a účet administrátora vytvoříme tak, že vytvoříme běžného uživatele a následně aplikujeme příkaz „net localgroup Administrators [uživatelské jméno] /add“, kterým přidáme uživatele do skupiny administrátorů.

2.6.5. Další praktické příkazy

Pokud se připojujeme z počítače, kde není IoTCoreWatcher, a potřebujeme zjistit jméno počítače za běhu aplikace, tedy v době, kdy není vypsáno na zobrazovací jednotce, můžeme si pomoci příkazem „hostname“, který nám vrátí jméno počítače, ke kterému jsme připojeni.

Mnoho ze základních síťových utilit je dostupných i ve Windows 10 IoT Core, jako např.: asping.exe, netstat.exe, netsh.exe, ipconfig.exe, nslookup.exe, tracert.exe, a arp.exe. Tyto nástroje se vyvolají napsáním jména a následně se s nimi pracuje obdobně jako v příkazové řádce či PowerShellu na stolních Windows.

Pro práci se soubory lze využít nástroje xcopy.exe, jehož možnosti vyvoláme příkazem „xcopy.exe /?“ . Příkaz se opět chová totožně jako na stolním počítači.

Pro zobrazení seznamu aktuálních startovních aplikací použijeme „startup.exe /d“. Pro přidání aplikace do automatického spuštění užijeme příkaz „start.exe / [Jméno][Příkaz]“. Nápoředu vyvoláme pomocí „startup.exe /?“ .

Obdobně jako u Linuxu je možné i Windows 10 IoT Core provozovat v grafickém režimu, nebo v režimu bez grafického výstupu. Tyto režimy se dají přepínat pomocí příkazu „setbootoption.exe [headed | headless]“, kde headed znamená grafický režim a headless znamená režim bez grafického výstupu. Tento příkaz není možné provozovat na žádném jiném systému z rodiny Windows 10.

K zobrazení aktuálních plánovaných úloh slouží příkaz „schtasks.exe“, který má oprávnění spustit jen administrátor. Pokud chceme do plánovaných úloh něco přidat, použijeme příkaz „schtasks.exe /Create“, pro okamžité spuštění určité úlohy „schtasks.exe /Run“ apod. Celý přehled možností získáme pomocí příkazu „schtasks.exe /?“.

Přístrojová utilita konzoly je užitečná v rozpoznávání a ovládání nainstalovaných zařízení a řidičů, a vyvolá se příkazem „devcon.exe“ doplněným o parametry, jejichž plný seznam zobrazíme příkazem „devcon.exe /?“. Příkaz nefunguje na ostatních systémech rodiny Windows 10.

2.6.6. Práce s registry OS

Pokud potřebujeme zpřístupnit registry, aby se zobrazily, nebo modifikovaly, je možné užít příkaz „reg.exe“ doplněný o parametry dle obrázku 21. Tento příkaz pracuje na všech zařízeních s Windows.

```
PS C:\Users\Jan> reg.exe /?
REG Operation [Parameter List]

Operation [ QUERY | ADD | DELETE | COPY |
             SAVE | LOAD | UNLOAD | RESTORE |
             COMPARE | EXPORT | IMPORT | FLAGS ]

Return Code: (Except for REG COMPARE)
0 - Successful
1 - Failed

For help on a specific operation type:
REG Operation /?

Examples:
REG QUERY /?
REG ADD /?
REG DELETE /?
REG COPY /?
REG SAVE /?
REG RESTORE /?
REG LOAD /?
REG UNLOAD /?
REG COMPARE /?
REG EXPORT /?
REG IMPORT /?
REG FLAGS /?
```

Obr. 21: Přehled parametrů příkazu „reg.exe“

2.6.7. Konfigurace služeb

Pro práci se službami systému máme několik možností. Úplně nejzákladnější výpis systémových služeb získáme pomocí příkazu „get-service“. Pro výpis aktuálně spuštěných systémových služeb můžeme užít příkaz „net start“. Pro spuštění služby můžeme užít například „net start jmeno_sluzby“ a pro zastavení služby „net stop jmeno_sluzby“. Další alternativou je užití nástroje sc.exe, jehož možnosti si vypíšeme příkazem „sc.exe /?“.

2.6.8. Úprava spouštění OS

Pokud bychom z nějakého důvodu potřebovali upravit spouštění systému, poslouží nám nástroj bcdedit.exe, jehož možnosti vyvoláme pomocí příkazu „bcdedit.exe /?“.

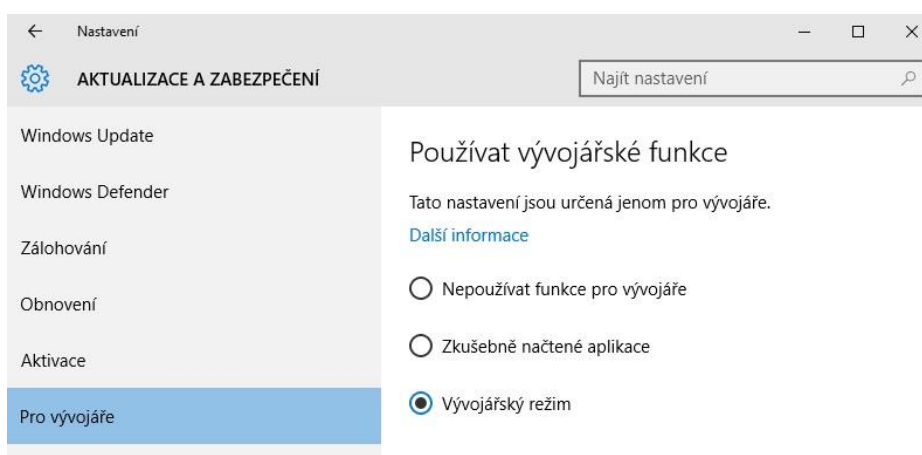
Konfigurační možnosti jsou obdobné jako u starších systémů Windows v souboru boot.ini a jsou ještě o něco rozšířené. Toto je již opravdu třešnička na dortu, protože jde o něco, co v případě Windows 10 IoT Core není příliš potřeba.

Protože je PowerShell velmi mocný nástroj pro práci s Windows 10 IoT Core, rozhodl jsem se v příloze A uvést přehled příkazů pro získání informací ze systému, tedy příkazů začínajících klíčovým slovem „get“.

2.7. Aplikace pro Windows 10 IoT Core

Jak jsem již zmínil, ve Windows 10 IoT Core se používají Windows Universal Apps. Psaní těchto aplikací je velmi blízké tzv. „Metro aplikacím“ pro Windows 8 a 8.1. Společnost Microsoft se rozhodla, že psaní těchto aplikací přiblíží programátorům formou velkého množství příkladů na GitHubu, kde je možné nalézt aplikace od úplně jednoduchých, jako je „Hallo world“, až po náročné projekty, jako je třeba domácí automatizace. Tyto příklady jsou doplněny o spoustu konferencí, jejichž záznamy lze nalézt i na serveru youtube.com. Chtěl bych upozornit, že tyto aplikace, ač mají hodně podobností s předešlou generací, jsou již opravdu univerzální a jsou odlišné od předešlé generace, i ve Visual Studiu se volí jiný typ projektu. Takovým na první pohled viditelným rozdílem je, že na rozdíl od předchozí generace nejsou tyto aplikace závislé na šablonách, jejich vzhled je již plně volitelný, ale kdo chce, může šablonu využít.

Nadále platí, že pokud chceme svoji aplikaci umístit do Store, musíme vlastnit ověřovací kód. Ověřovací kód mohou studenti získat na Dreamsparku¹, kde je možné si stáhnout další pomoc v podobě GitHub Student Developer Pack a pro Visual Studio 2015 i Xamarin for Students. Ověřovací kód bohužel potřebujeme v případě, že aplikaci chceme užít ve Windows 10 IoT Core, ač tento systém nemá přístup do Store. Ověřovací kód nepotřebujeme v případě, kdy aplikace nasazujeme na počítače, které pracují v režimu „Zkušebně načtené aplikace“, nebo „Vývojářský režim“. Na počítači, kde jsou vyvíjeny aplikace, bych doporučoval mít zapnutý jeden z těchto režimů, ideálně „Vývojářský režim“, kde se dá pracovat i s operačním systémem jako takovým. Tyto režimy je možné zvolit v nastavení, tedy cestou start => nastavení => aktualizace a zabezpečení => Pro vývojáře (viz obrázek 22).

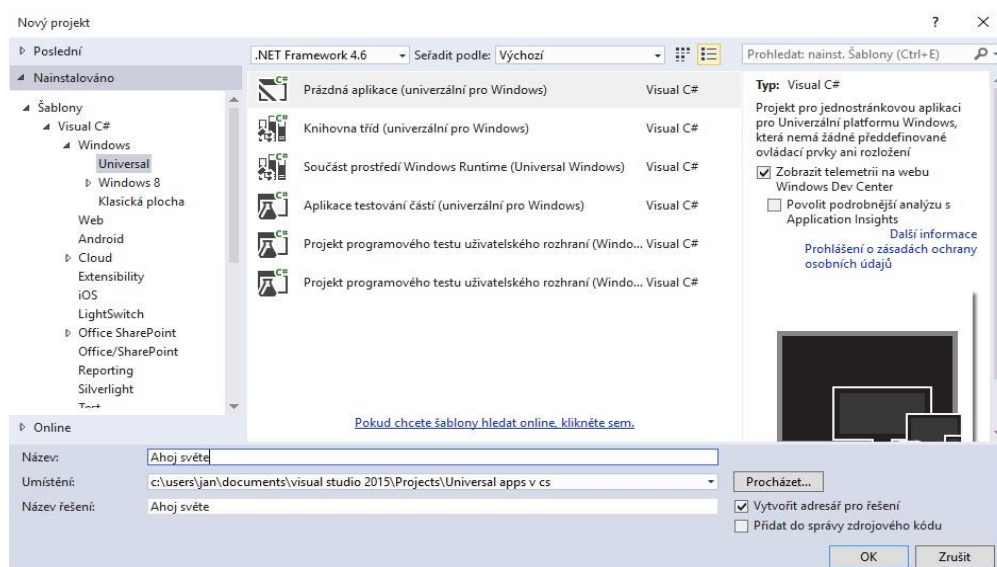


Obr. 22: Aktivace vývojářského režimu v systému Windows 10

Nyní si ukážeme, jak správně založit projekt Windows Universal Apps a vytvořit aplikaci pracující na Windows 10 IoT Core. Na projektu aplikace „Ahoj lidi“ si ukážeme vše podstatné kolem ovládání Visual Studia 2015, které budeme potřebovat i pro složitější aplikace běžící na Windows 10 IoT Core.

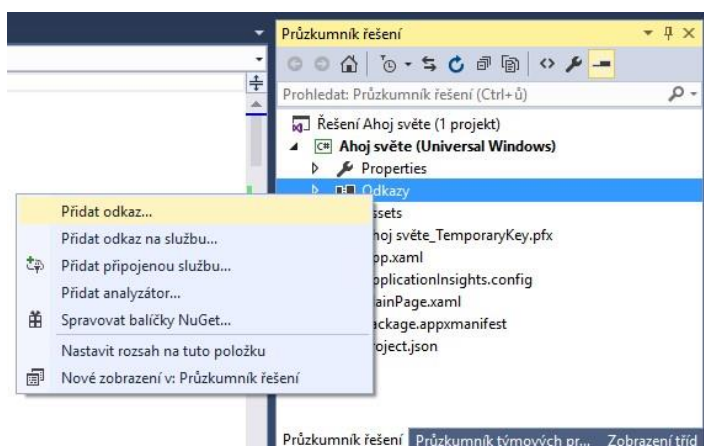
Při vytvoření nového projektu Visual Studia si zvolíme jazyk (C++, nebo C#), ten rozklikneme, vybereme Windows => Universal a položku prázdná aplikace (viz obr. 23). Následně si pojmenujeme aplikaci a zvolíme si adresář, do kterého se projekt uloží. Net Framework můžeme nechat výchozí, nebo využít verzi 4.6.

¹ Na MSDN AA, který vlastní škola registrovaná do programu (střední či vysoká), nebo na adrese: <https://www.dreamspark.com/Student/Software-Catalog.aspx>, kde je uveden jako Windows Store Developer Account.



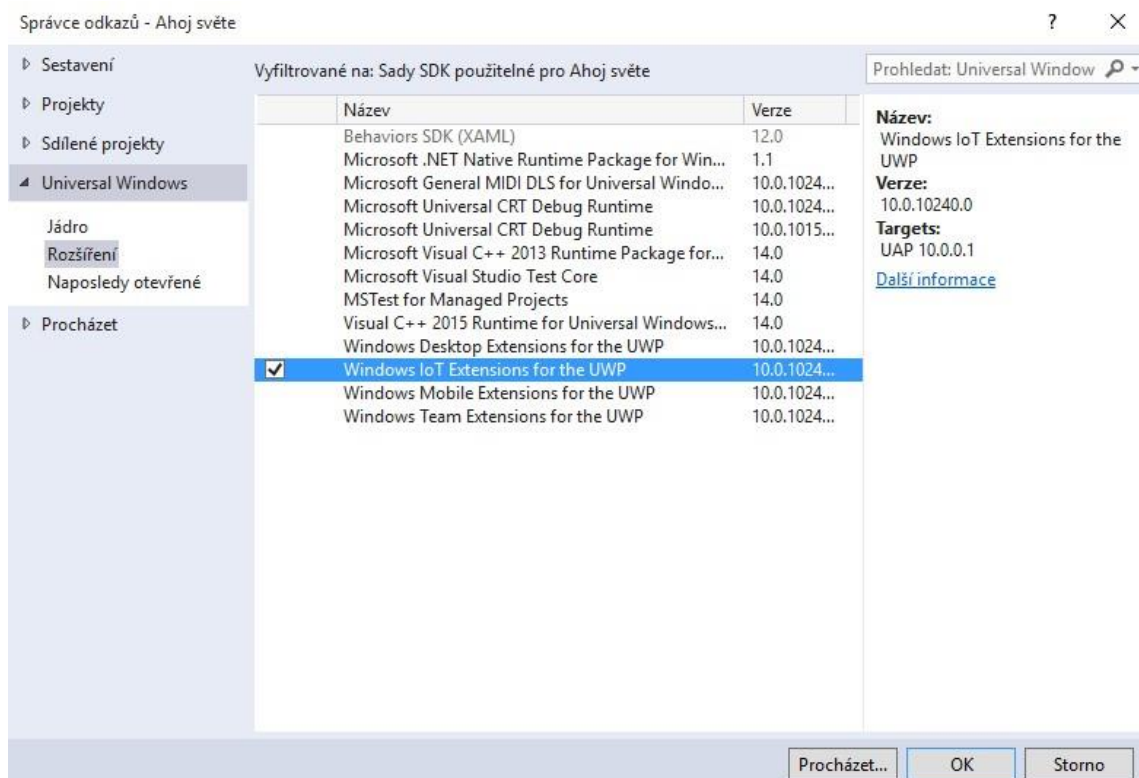
Obr. 23: Založení projektu Windows Universal Apps ve Visual Studiu

Pokud máme nastaveno, vše potvrdíme tlačítkem OK a počkáme, až Visual Studio vytvoří požadovaný projekt. Nežli se pustíme do vlastní práce na projektu, je potřeba ještě připojit reference, pokud se bude jednat o aplikaci pracující s GPIO sběrnici, nebo aplikaci, u které víme, že bude využívat senzory v telefonech či tabletech. Přidání těchto referencí provedeme kliknutím pravým tlačítkem na položku Odkazy v sekci Průzkumník řešení a zvolíme „Přidat odkaz“.



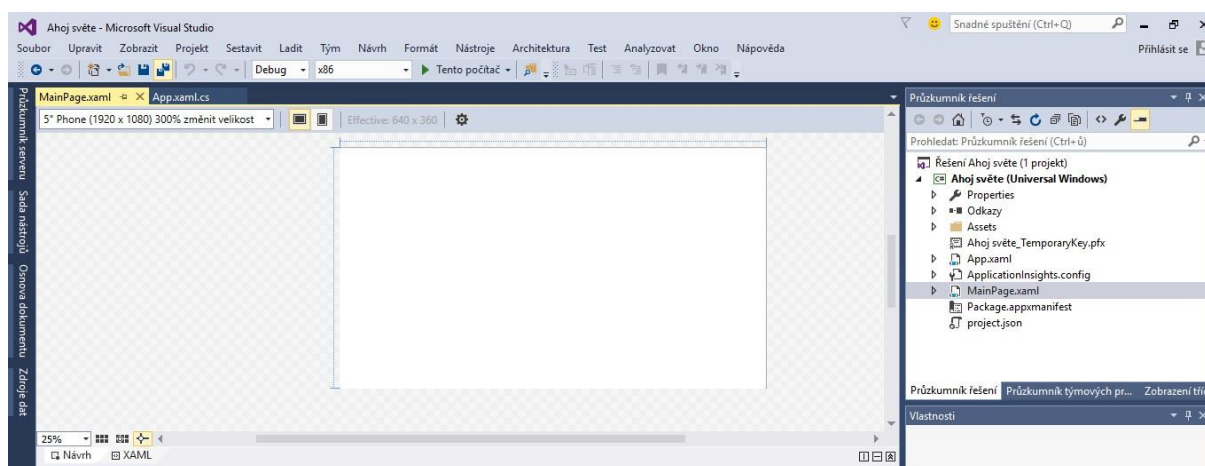
Obr. 24: Postup přidání referencí do projektu Visual Studia

V otevřeném dialogovém okně se přepneme v levém sloupci z položky Jádno na položku Rozšíření a zaškrtneme Windows IoT Extensions for the UWP. Na psaní pro Raspberry žádnou další rozšiřující knihovnu nepotřebujeme, pokud to nevyžaduje náš algoritmus, pak již musíme vycházet ze znalostních databází na webu. Všechno potvrdíme tlačítkem OK.



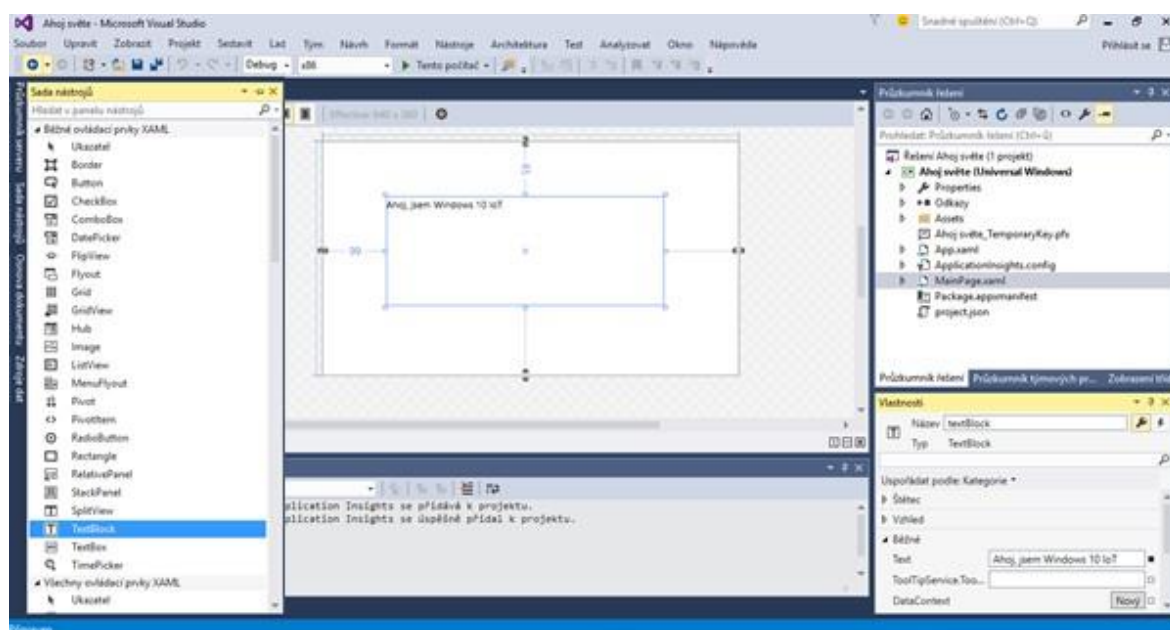
Obr. 25: Přidání referencí pro Windows IoT

Nyní máme již všechno připraveno k tomu, abychom mohli začít programovat. Jak jsem slíbil, ukážeme si jednoduchou aplikaci „Ahoj světe“, která vypíše text na monitor. Grafickou podobu aplikace vytvoříme buď v podobě příkazů v jazyce Xaml, nebo grafickým návrhem (nelze pod Win 8.1). Ukážeme si druhou cestu, takže si dvakrát klikneme v Průzkumníku řešení na MainPage.xaml a počkáme na načtení editoru.



Obr. 26: Grafický editor souboru MainPage.xaml

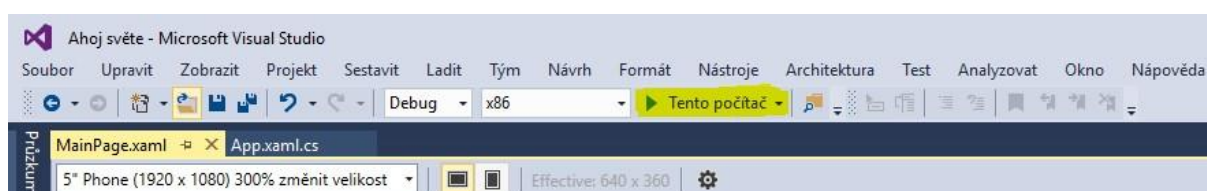
Zde si otevřeme sadu nástrojů (po levé straně) a zvolíme si TextBlock a jako v malování si vybereme oblast na „plátně“, kam chceme text umístit.



Obr. 27: Vložení textového bloku v návrhu MainPage

Vpravo dole (pod Průzkumníkem řešení) nalezneme okno Vlastností, kde vyplníme v záložce Běžné políčko textu, který chceme zobrazit. Dále položku Běžné sbalíme kliknutím na šipku a otevřeme si záložku Text. Zde si zvolíme velikost, řez písma, zarovnání apod. Nezapomeneme celý projekt uložit kliknutím na dvojici disket (nepoužijeme tentokrát jednu disketu, nebo **ctrl+s**, protože by se nám uložily změny jen v daném souboru).

Nyní si můžeme naši aplikaci vyzkoušet na lokálním počítači, pokud jsme nikde neudělali chybu. Aplikaci spustíme pomocí zelené šipky nad editorem (viz obrázek).



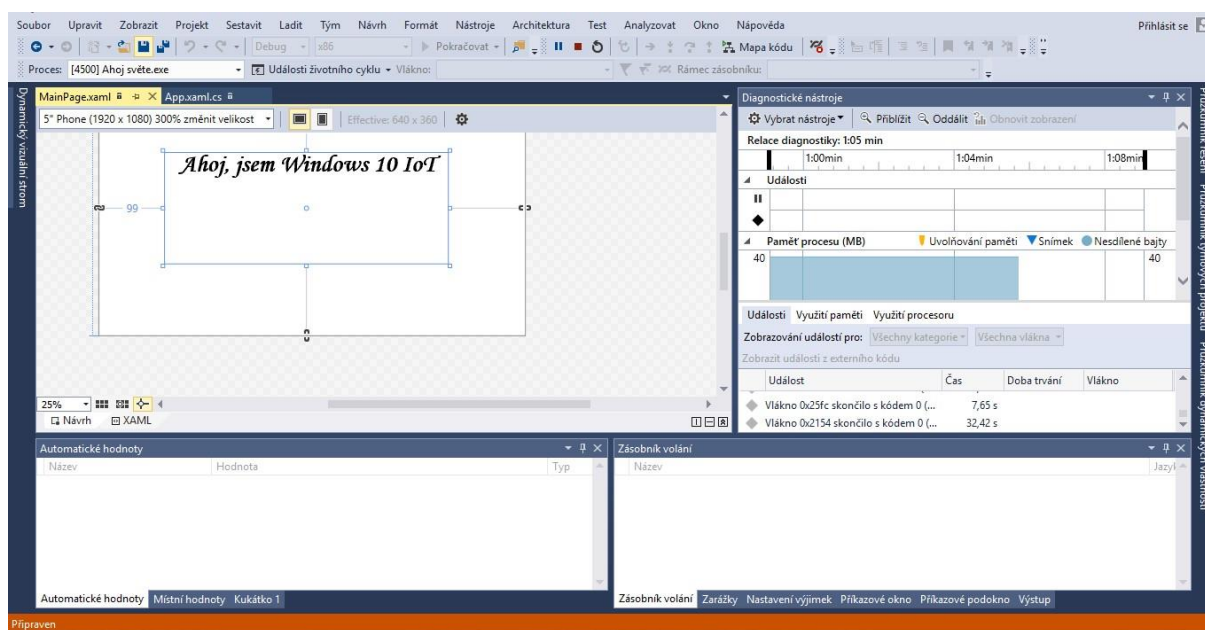
Obr. 28: Spuštění ladění ve Visual Studiu

Pro otestování aplikace na lokálním počítači necháváme výchozí nastavení ladění (viz obrázek), nebo jej můžeme přepnout rozkliknutím do režimu x64. Úspěšným laděním zároveň dojde k instalaci aplikace na náš lokální počítač a nalezneme ji ve startu mezi všemi aplikacemi, pokud máme systém nastavený v jednom ze dvou výše zmiňovaných režimů. Podotýkám, že u první kompilace na danou platformu (x64, nebo x86) trvá několik minut, nežli dojde k sestavení aplikace, protože Visual Studio si ještě instaluje podpůrné komponenty. Pokud vše provedeme správně, měly bychom získat aplikaci a rozložení Visual Studia dle následujících obrázků.



Ahoj, jsem Windows 10 IoT

Obr. 29: Aplikace Ahoj světe při ladění

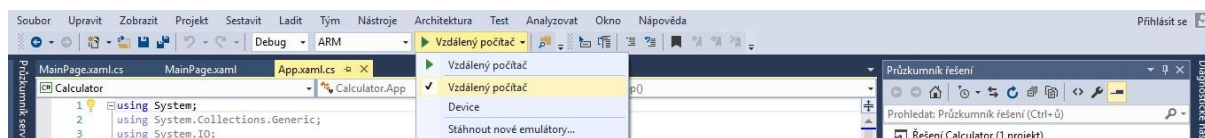


Obr. 30: Visual Studio při ladění aplikace Ahoj světe

2.7.1. Vzdálené ladění kódu

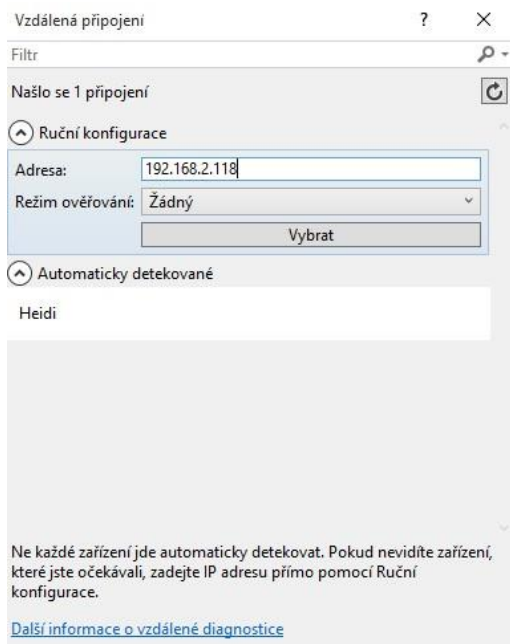
Před spuštěním vzdáleného ladění kódu je dobré si ověřit, že v systému běží „aplikace“ ZwaveHeadlessAdapterApp, což je aplikace zajišťující vzdálené ladění kódu z Visual Studia. Tato aplikace se automaticky spouští se startem systému a obvykle není ukončována, jediná možnost jak ji ukončit, je chyba uživatele, který ji vypne pomocí webového prostředí, nebo pomocí PowerShellu. Pokud tato aplikace neběží, Visual Studio odpoví chybou nedostupného vzdáleného zařízení.

Prvním krokem je přepnutí Visual Studia do ARM (v případě MinnowBoard Max x86) a následně si rozklikneme nabídku u šipky, kterou spouštíme ladění, a zvolíme si vzdálený počítač (viz. Obr. 31).

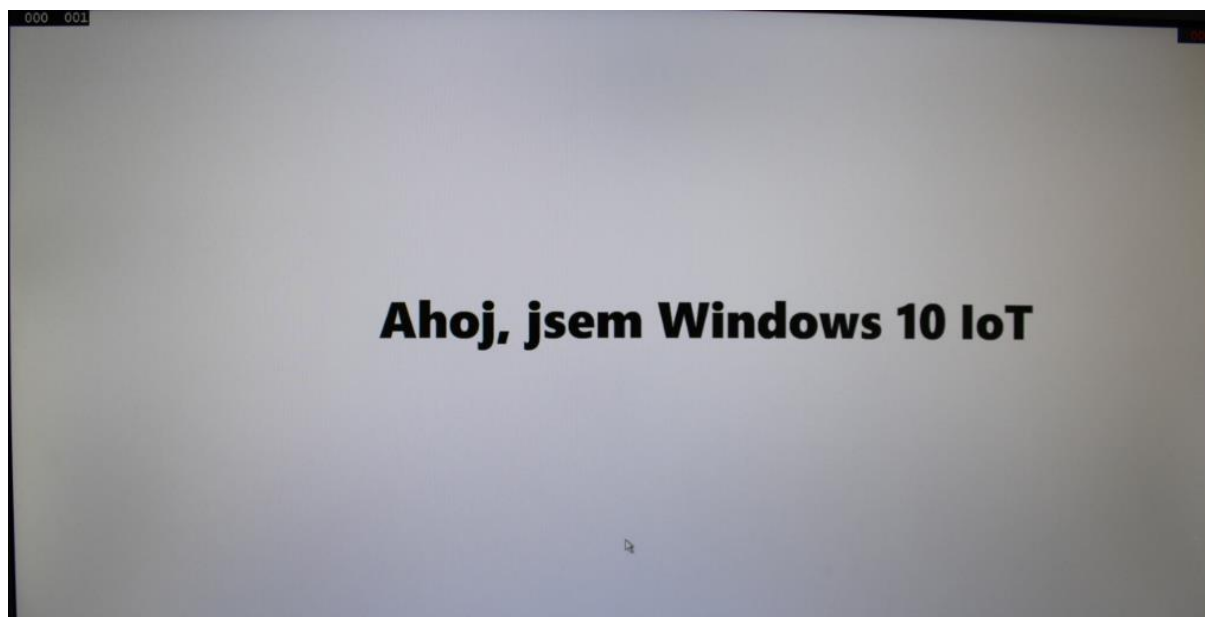


Obr. 31: Nastavení režimu vzdáleného ladění

Následně si nastavíme vzdálené zařízení, na kterém bude probíhat ladění. Vyplníme IP adresu (verze 4) a přepneme režim z „Windows“ do „Žádné“, jak je vidět na obrázku 32, a všechno potvrdíme tlačítkem „Vybrat“. Nyní je již všechno připraveno a můžeme spustit ladění stejně, jako na lokálním počítači.



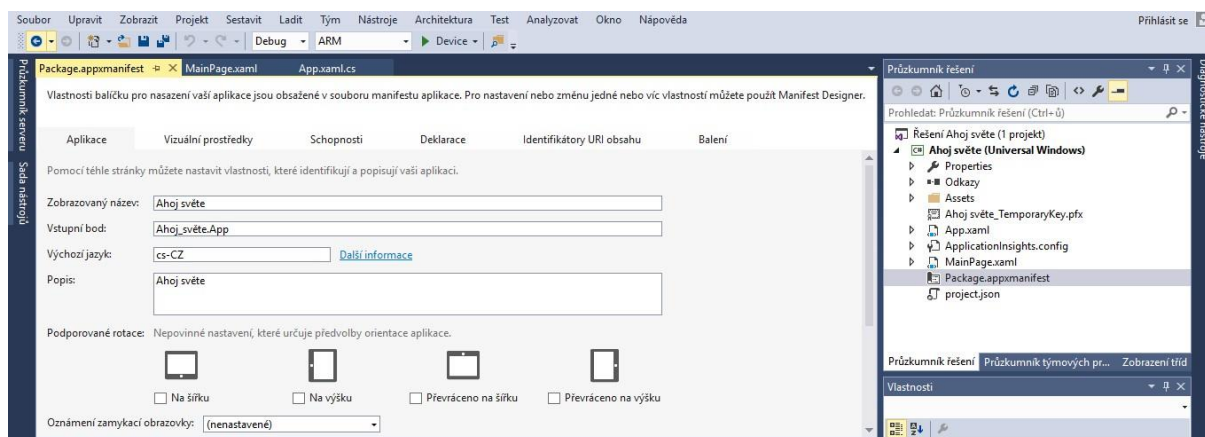
Obr. 32: Nastavení vzdáleného ladění



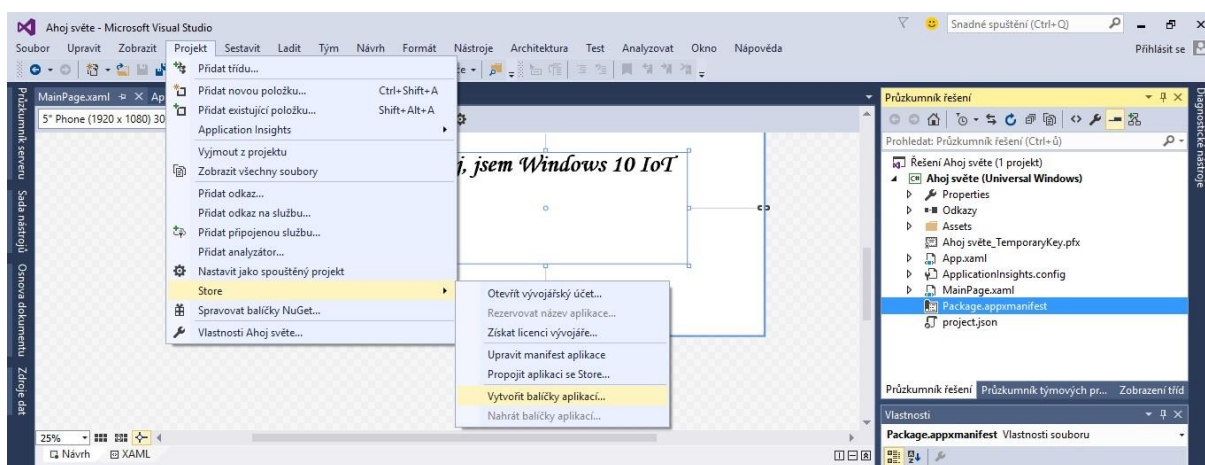
Obr. 33: Vzdálené ladění na zobrazovači Raspberry Pi

2.7.2. Vytvoření balíčku aplikace

Aplikaci máme napsanou a odladěnou, přišel čas vytvořit instalační balíček, který nainstalujeme do našeho Windows 10 IoT Core. Pro nahrání programu potřebujeme mít aplikaci ve formátu appx. Pokud chceme aplikaci blíže nastavit pro umístění do Store, provedeme nastavení pomocí dvojkliku na Package.appxmanifest (viz obrázek) a jednoduchou konfigurací. Balíček aplikace získáme z Visual Studia velmi jednoduše (pokud máme vývojářský účet), stačí jen kliknout na nabídku Projekt, zvolit Store a vytvořit balíčky aplikací.



Obr. 34: Konfigurace aplikace pro umístění do Store



Obr. 35: Otevření průvodce pro tvorbu balíčku Windows universal Apps

Pokud opravdu chceme aplikaci použít jen pro Windows 10 IoT Core, průvodce nakonfiguruje dle obrázků.

Vytvořit balíčky aplikací

Vytvořit balíčky

Chcete sestavit balíčky pro nahrání na Windows Store?

☒ Ano
☐ Ne

Sada Visual Studio stáhne informace potřebné k odeslání balíčků na Store. Balíčky je také možné používat v počítači s nainstalovanou vývojářskou licencí nebo v počítači, který splňuje požadavky pro zkušební načtení aplikací pro Windows Store před prodejem. Další informace viz [vývojářská licence](#) a požadavky pro [zkušební načtení před prodejem](#). K webu Windows Store je nutné se přihlásit pomocí účtu Microsoft. Pokud tento účet ještě nemáte, můžete si jej [vytvořit](#).

Předchozí Další Storno

Obr. 36: Vytvoření balíčku aplikace krok 1

Vytvořit balíčky aplikací

Vybrat a nakonfigurovat balíčky

Výstupní umístění:

c:\users\jan\documents\visual studio 2015\Projects\Universal apps v cs\Ahoj světe\Ahoj světe\AppPackages\ ...

Verze:

1 . 0 . 0 . 0

☒ Automatický přírůstek

Generovat svazek aplikací:

Vždy

[Co znamená označení „svazek aplikací“?](#)

Vyberte balíčky, které chcete vytvořit, a mapování konfigurace řešení:

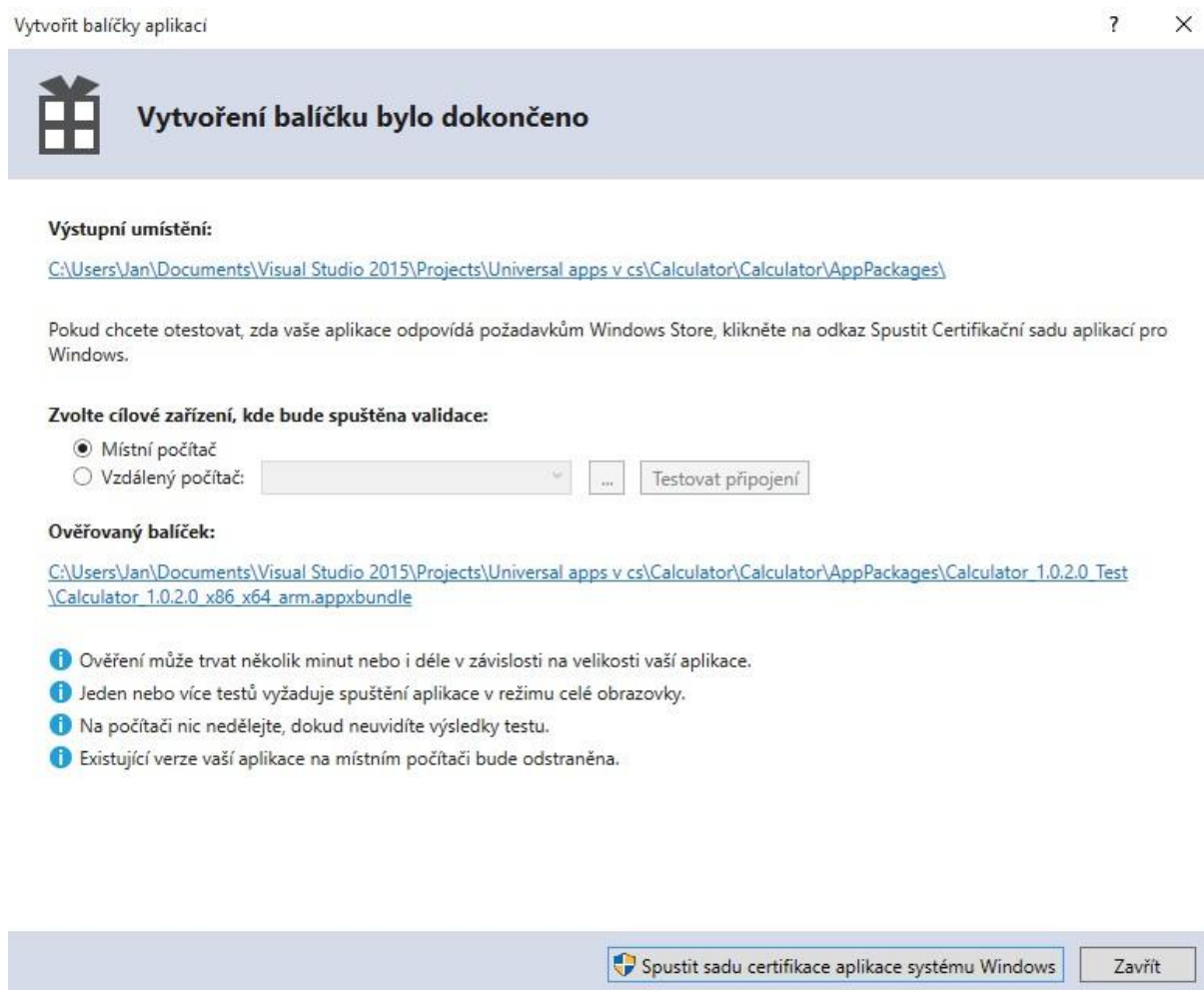
Architektura	Konfigurace řešení
☐ Neutral	Žádný
☐ x86	Release (x86)
☐ x64	Release (x64)
☒ ARM	Release (ARM)

☒ Zahnout úplné soubory symbolů PDB pro účely analýzy chyb aplikace [Další informace](#)

Předchozí Vytvořit Storno

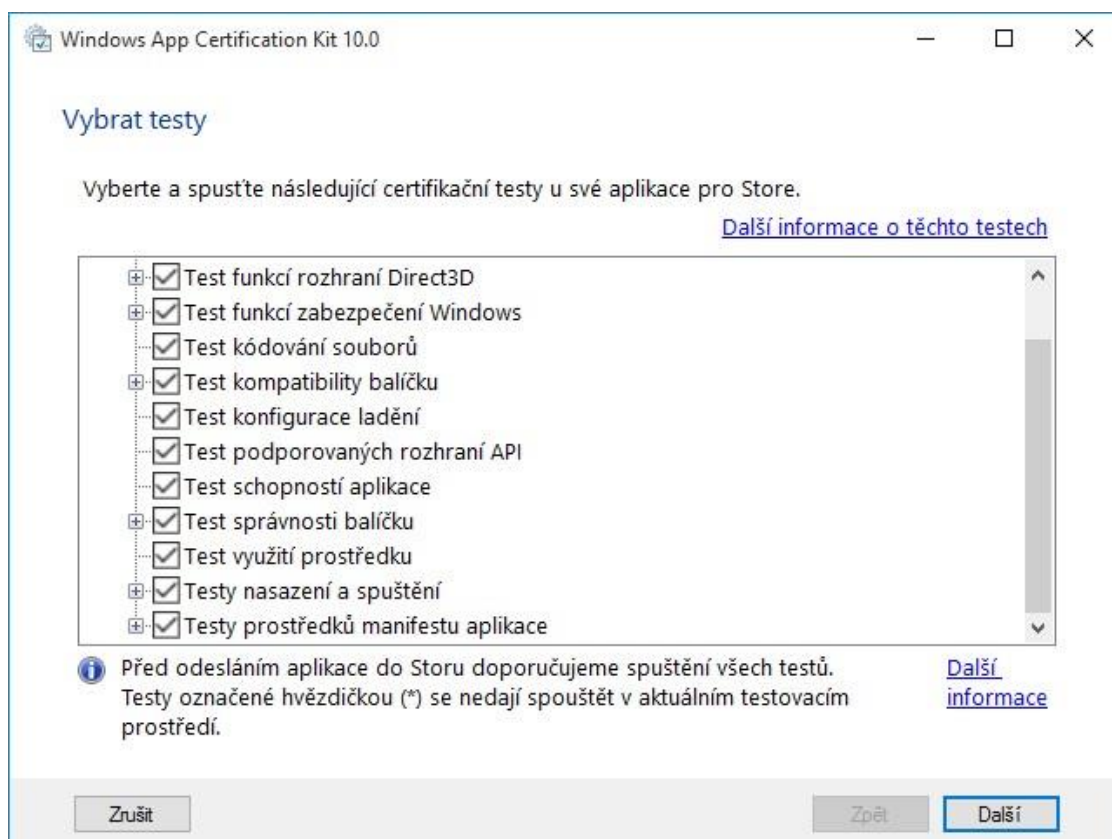
Obr. 37: Vytvoření balíčku aplikace krok 2

V kroku dva si zvolíme architekturu (můžeme klidně ponechat všechny), vyplníme verzi a nastavíme typ konfigurace řešení. Nic nám nebrání ponechat původní nastavení na Debug, ale platí tady, že konfigurace řešení musí být jednotná pro všechny vytvářené architektury. Všechno završíme kliknutím na tlačítko Vytvořit. Po dokončení procesu se nám zobrazí tabulka pro vytvoření certifikátu k aplikaci (pokud se jedná o konfiguraci řešení Release).



Obr. 38: Vytvoření certifikátu aplikace

Při vytváření certifikátu dojde k otevření sady automatizovaných testů, které prověří aplikaci, zda splňuje podmínky Store. Doporučuji provést všechny testy, pokud aplikace zůstává pouze na našem Windows 10 IoT Core, můžeme si dovolit zrušit některé testy, o kterých víme, že je nebudeme potřebovat.



Obr. 39: Nastavení automatizovaných testů aplikace při generování certifikátu

Po dokončení testů budeme mít k dispozici kompletní výsledky a v případě hodnocení „Neúspěšné“ budeme mít popis chyby a nástin možného řešení chyby. K tomuto html souboru je možné se kdykoliv vrátit, protože je uložen v adresáři projektu. Pokud zvolíme jako konfiguraci řešení Debug, testy se nám nenabídnou, balíčky jsou včetně certifikátů vygenerovány rovnou.

2.7.3. Přímá instalace pomocí Visual Studia

K tomuto způsobu nebudeme potřebovat vývojářský účet. Jediné, co je potřeba, je Visual Studio a upravit pár vlastností projektu před tím, než začneme ladit na Windows IoT. Otevřeme si vlastnosti projektu, kde na záložce „Balení“ změním název balíčku z předdefinovaného na vlastní, který je shodný s názvem aplikace. Následně stačí jen nastavit vzdálené zařízení, stejně jako tomu bylo při ladění aplikace.

Po spuštění ladění počkáme, až se nám aplikace spustí, a vyzkoušíme všechny její funkce. Pokud aplikace funguje, jak má, přejdeme do administrace systému pomocí webového prohlížeče a přejdeme na záložku Apps, kde uvidíme ve spuštěných aplikacích aktuálně běžící aplikaci. Otevřeme si seznam všech aplikací, kde nám aplikace běží, vybereme ji a klikneme na tlačítko

„Set Default“. Pokud se náš systém dotáže na restart (může se lišit v závislosti na použitém Buildu), tak se zobrazeným hlášením ukončíme Debug ve Visual Studiu a povolíme restart pomocí webového rozhraní.

Po dokončení restartu by mělo dojít k načtení aplikace, kterou jsme si instalovali, a doporučuji opět aplikaci vyzkoušet. Když jsme si jistí, že aplikace pracuje, jak má, tak si můžeme pomocí webového rozhraní změnit výchozí aplikaci zpět na původní, pokud to tak chceme, naše nově přidaná aplikace již zůstane uložena v zařízení.

Tento postup má tu výhodu, že nepotřebujeme vlastnit vývojářský účet a vše nám proběhne na naší lokální síti. Další výhodou je, že se nezdržujeme nasazením testů aplikace a nemusíme tudíž řešit, jestli jsme v balíčku náhodou nezapomněli certifikát. Nevýhodou tohoto postupu je, že pomocí Visual Studia jej musíme opakovat při každé reinstalaci a také při nasazování na další zařízení. Tento postup je vhodný pro samostatné vývojáře, pracující samostatně na volné noze, nebo pro kutily či studenty.

2.7.4. Aplikace v jazyce Python

Systém Windows 10 IoT Core podporuje kromě Windows Universal Apps také možnost psát aplikace v jazyce Python, což je interpretovaný jazyk. Dle oficiálních zdrojů je Python určen k psaní Core aplikací, tedy aplikací bez grafického uživatelského rozhraní.

Pro psaní aplikací v tomto jazyce je vyžadováno Visual Studio 2015 update 2 s doplňky pro vývoj v Pythonu, Windows 10 IoT Core verze 10.0.10586.0 a novější, Python verze 3.x, Windows SDK verze 10586 a Python UWP SDK (zatím pouze Alfa verze²).

Při zakládání projektu volíme v sekci ostatní jazyky Python a typ projektu Windows IoT, další práce s Visual Studií je obdobná jako v jiných jazycích.

Pro ladění na vzdáleném zařízení platí následující postup. Opět jako v předešlém případě zvolíme architekturu v závislosti na jednodeskovém počítači a režim vzdáleného ladění. Následně klikneme pravým tlačítkem v průzkumníku řešení na název projektu a z kontextového menu zvolíme vlastnosti. V UWP nastavíme jméno zařízení (nebo jeho IP verze 4) a číslo portu na 5678. Jako ve kterémkoli jiném případě můžeme začít ladit.

² Stav ke dni 24.7.2016 je Python UWP SDK verze 1.3 Alfa

Hotovou aplikaci je možné nasadit s využitím Visual Studia velice jednoduše, stačí pouze přepnout režim z Debugg na Release a opakovat postup pro vzdálené ladění.

2.7.5. Ladění a nasazení UWP v C++

Vytvoření projektu a jeho vývoj je obdobný jako C# a VB (viz kapitola 3.5.1), ale vzdálené ladění na zařízení s Windows 10 IoT Core, se liší. V C++ je potřeba nastavit režim, architekturu systému a vzdálené zařízení. Následně v průzkumníku řešení (Solution Explorer) klikneme na název projektu s ikonou jazyka (tučně) pravým tlačítkem myši a zvolíme vlastnosti. Ve vlastnostech si otevřeme vlastnosti konfigurace (druhé odshora) a otevřeme záložku ladění. Zkontrolujeme, případně opravíme, režim debuggeru, který by měl být vzdálené zařízení. Následně vyplníme jméno zařízení (případně IP adresu verze 4) do položky jméno zařízení a jako typ ověření zvolíme univerzální nešifrovaný protokol. Nyní je vše připraveno ke spuštění ladění a případné instalaci aplikace pomocí Visual Studia, jak je popsáno v kapitole 2.7.3, případně k následné tvorbě balíčků (viz kapitola 2.7.2).

2.8. Srovnání vývoje mezi verzemi 10531 a 14342

Za 7 měsíců vývoje, tedy od verze 10.0.10531 k verzi 10.0.14342.1000 odvedl Microsoft opravdu znatelný kus práce. Systém se dočkal mnoha změn a novinek, které opravdu výrazně zjednodušují uživatelům systému život. Výrazných změn nedosáhl jen systém jako takový, ale projevy se i v řídicích aplikacích pro stolní počítač. Nyní si popíšeme postupně změny od právě řídicích aplikací a proces instalace až po novinky, které poskytují vlastní systém.

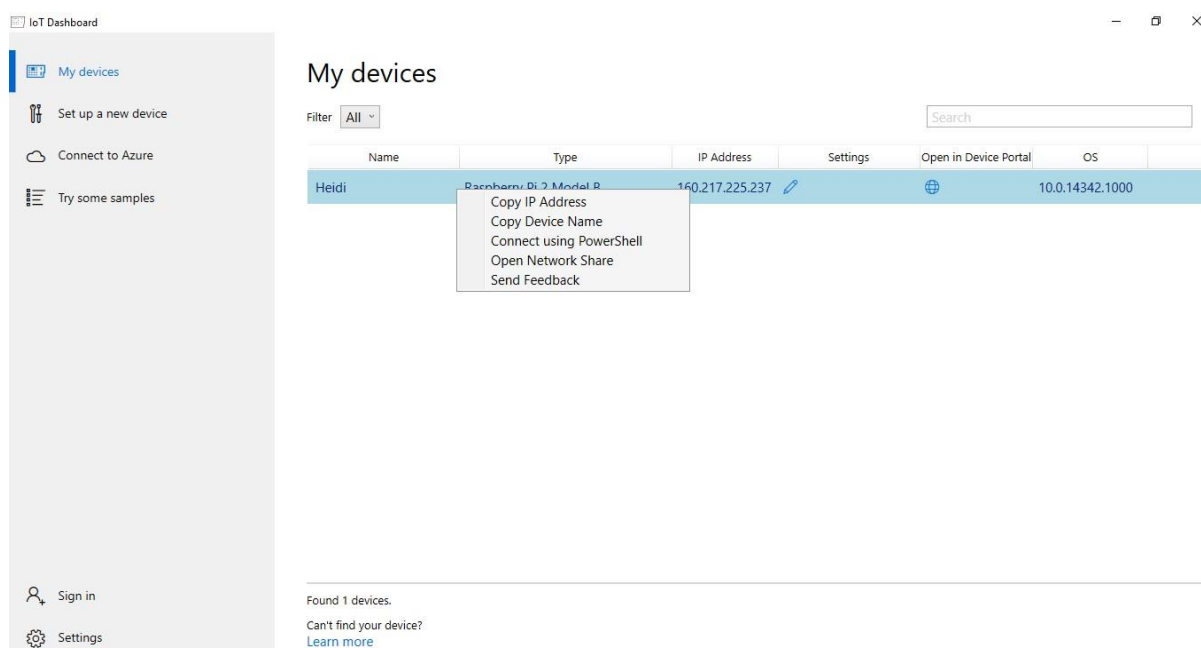
2.8.1. Novinky v řídicích aplikacích pro stolní počítač

Zde se Microsoft rozhodl uživatelům výrazně zjednodušit práci vydáním aplikace Windows 10 IoT Core Dashboard, která slučuje a aktualizuje předcházející dvojici aplikací IoT Image Helper a Windows IoT Core Watcher, který od verze 10.0.14342.1000 zcela nahrazuje. Zde je potřeba pochválit Microsoft za celkové zpracování aplikace, které převzalo filozofii z aplikace Server Manager Dashboard. Další kladné body z pohledu uživatele je nutné Microsoftu připsat za integrovaný proces instalace systému, který funguje v aplikaci Windows Recovery Tool pro mobilní telefony. Též integrace Azure je praktickým krokem stejně jako

možnost přímé instalace zkušebního příkladu aplikace do zařízení bez nutnosti spuštění Visual Studia.

Poslední z výčtu zlepšení, které se podařilo nové aplikaci dosáhnout, je zpráva připojení, kde se uživatelé dočkali přímého vyvolání PowerShell relace, či přímého vyvolání nastavení sdílení sítě.

Abych jen nechválil, tak i novou aplikaci provází některé neduhy. Prvním neduhem je, že aplikace je oficiálně podporovaná jen pro Windows 8 a novější (nepodporuje serverové verze systému), dalším neduhem jsou časté požadavky na připojení k internetu, které mohou snižovat komfort při práci s aplikací v uzavřené interní síti. Posledním neduhem, který trochu s touto aplikací souvisí, je fakt, že nové verze systému již nedokážou komunikovat s aplikací Windows IoT Core Watcher, což uživatele nutí přejít k nové aplikaci, i když používají Image Helper a stahují si instalátory v podobě ISO souborů.

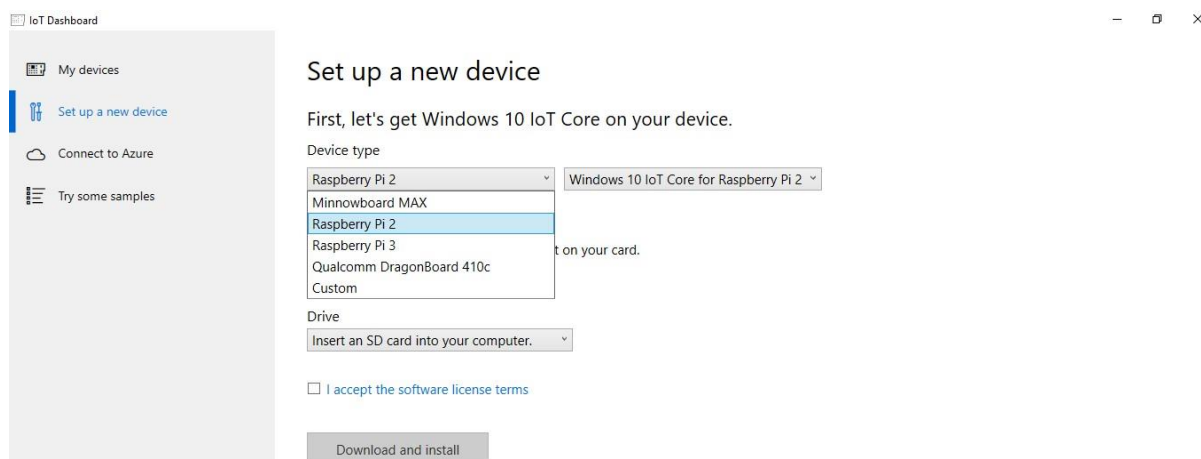


Obr. 40: Možnosti připojení se k Windows 10 IoT v aplikaci IoT Dashboard

2.8.2. Novinky v instalaci systému

Novinkou je možnost použít aplikaci IoT Dashboard, která umožňuje dva způsoby instalace. Jedním je volba Custom, která vede na stejný způsob práce jako s Windows IoT Image Helper a je určena pro instalaci buildů Insider, případně starších buildů systému pomocí souboru ffu. Novinkou je možnost přímé instalace aktuální verze release větve systému online obdobně,

jako je tomu u telefonu s WP. Rozdílem proti telefonům je, že musíme zvolit typ desky, pro kterou chceme připravit paměťovou kartu se systémem.



Obr. 41: Instalace systému pomocí IoT Dashboardu

2.8.3. Novinky v systému pro build 10.0.14342.1000

Nejen na instalaci a řídicí aplikaci v Redmondu zapracovali, i systém se dočkal opravdu velkého množství novinek, odbourání mnohých problémů i kosmetických změn designu řídicí webové stránky. V této části si ukážeme některé novinky, kterých se systém dočkal.

2.8.3.1. Novinky ve webovém rozhraní pro správu OS

Na domovské stránce přibyla možnost změny časového pásma, nastavení rozlišení a obnovovací frekvence zobrazovací jednotky a její orientace. Na stránce Apps se kromě změny podoby seznamu všech instalovaných aplikací objevila i možnost odstranění aplikačního balíčku a možnost nastavení startup chování aplikace.

Znatelných změn doznala i stránka Device Manager, která mimo změny designu do dalece přehlednější varianty, byla rozšířena o možnost aktualizace ovladačů, což je umožněno začleněním systému do služby Windows Update.

2.8.3.2. Nové funkce integrované do systému

Úplně novou funkcionalitou je průzkumník souborů, který nápadně připomíná první oficiální průzkumník souborů pro Windows Phone 8.

Novinkou v oblasti sítí je IoT Onboarding, což je integrace často vytvářené aplikace. Jde o sdílení připojení k internetu pomocí Wi-Fi, kdy IoT systém můžeme využít jako přístupový bod.

Mezi další novinky, které jen krátce zmíním, patří Windows Update, TPM 2.0, které je možné nastavit pro konkrétní sběrnici a Windows 10 IoT Remote Server, který umožní sdílení obrazového výstupu a interakcí s aplikacemi pomocí klientské aplikace, kterou je možné stáhnout pomocí aplikace Store do počítačů, tabletů a telefonů s Windows Phone, či Windows 10 Mobile. Tuto možnost musíme povolit na stránce Remote.

Pokud jde o požadavky pro běh systému, ty zůstaly stejné, jen je potřeba počítat s tím, že nová verze systému zabere na kartě více místa nežli náš původní starý systém, s čím je potřeba počítat při provádění upgrade a zvážit, zda se nám budou všechna zálohovaná data vejít zpět na kartu, nebo je potřeba instalovat systém již na větší kartu.

3. Testování systému

3.1. Strategie testování

Celkově strategie testování vychází z postupů exploratorního testování. Tento přístup byl zvolen jako nejlepší možný z důvodu, že jediné, co je k dispozici, je již instalační médium a zprovozněný OS, proto není možné případná slabá místa odhalit analýzou zdrojového kódu. Přístup testování nazývaný „Black box“ zde není vhodný, protože systém je obecným OS, nejde o specializovaný systém, jako jsou například bankovní systémy. Při exploratorním testování dochází k seznámení testera s testovaným produktem v jeho průběhu, což je můj případ. Exploratorní testování s sebou nese riziko opomenutí otestování nějaké komponenty a nevýhodu v minimální možnosti použití automatizace. Tento druh testování je poměrně hodně závislý na zkušenostech a osobě testera a často bývá zaměňován s Adhock testováním. Exploratorní testování je dobré pro první fáze testování již ucelených systémů (především operačních) a testování uživatelských rozhraní, umožňuje poměrně jednoduše určit nejčastěji používané komponenty a odhalit, zda se otestovalo opravdu všechno a navrhnout, co je potřeba blíže otestovat tak, aby se nic nevynechalo.

Windows 10 IoT Core je specializovaný IoT operační systém, že je potřeba se při testování zaměřit na specializované funkce, jako je práce s GPIO a I²C sběrnicemi, nicméně to do velké míry ovlivňuje obslužná aplikace, což je potřeba vzít v potaz. Dále je pak potřeba se zaměřit na vzdálené řízení systému (nezávisle na běžící aplikaci), ale také na dostupnost aplikace spuštěné na systému v případě její chyby či chyby OS nebo HW, tedy jak rychle je systém schopen po kolapsu provést pokus o obnovu běhu aplikace a k jaké škodě při tomto obnovovacím procesu dojde.

Od IoT OS obecně očekáváme jednoduché a spolehlivé vzdálené ovládání, stabilitu běhu a nějaký automatizovaný postup reakce na chybu aplikace, vnitřní chybu systému či chybu HW, aby byla nejvyšší možná dostupnost spuštěné aplikace. Dále pak očekáváme vyřešení práce se specializovanými sběrnicemi, které nám konkrétní průmyslový počítač zpřístupňuje.

Strategie je poměrně jednoduchá, nicméně umožňuje dobře určit, co je otázkou systému a co již je otázkou aplikace, u které je snaha se důsledně držet pokynů Microsoftu, aby aplikace jako taková co nejméně ovlivňovala výkon celku.

Prvním bodem je tedy zjištění sledovaných parametrů při spuštění na prázdno a ustálení systému (špička a ustálený stav), v systému běží jen nativní aplikace pro vzdálený přístup, základní aplikace systému a interní aplikace pro sběr dat.

Další důležitou částí testování je sledování parametrů při ladění (vzdálený debugg) různě náročných vzorových aplikací, protože jde o poměrně častou činnost jednak při vývoji samotných aplikací, při hledání příčin jejich nestandardního chování.

Neméně důležitým bodem testování je sledování parametrů při běhu různě náročných aplikací, aby bylo možné posoudit, na kolik je systém pružný a jak reaguje na různou míru zatížení.

Vzhledem ke specializaci systému je potřeba prověřit pomocí nejjednodušších možných aplikací práci se specializovanými sběrnici a rychlost reakcí systému na události generované sběrnici.

Chování systému jsem dále prověřil pomocí záměrně zacyklené aplikace, která neustále zabírala exponenciálním trendem paměť a udržovala procesor v zatížení nad 50 %. Cílem bylo zjistit, kde se nachází hranice kdy, již systém zasáhne a aplikaci ukončí.

Důležitým bodem je sledování kritických chyb, v případě tohoto typu systému především BSOD a také toho, zda se systém dokáže z BSOD probít, či nikoliv, s čímž souvisí i monitoring stability aktualizací služby Windows update.

Poslední bod testování je výrazně nestandardní a spočívá ve sledování systému při dlouhodobém běhu aplikací, řešící některý z tradičních či méně běžných IoT scénářů. Tento test má simulovat reálné nasazení systému s konkrétním HW a není zde cílem sledovat spotřebu prostředků, ale plynulost a stabilitu celku tvořeného aplikací a systémem. Opravdovým cílem je prokázat, že systém je (či není) vhodný pro reálné nasazení a zda zvládne to, co jeho již dlouhá léta rozvíjený Linuxoví konkurenti v oblasti ryziho IoT.

3.1.1. Sledované parametry

U většiny testů jsem sledoval spotřebu systémových prostředků, tedy využití RAM a CPU. Dále se sledovala stabilita systému, evidovaly se veškerá zamrznutí a pády systému. Při specifických testech IoT scénářů jsem sledoval odezvu systému na vnější událost a také jsem sledoval fyzicky pomocí osciloskopu signály jdoucí na a z GPIO.

Dále jsem sledoval dobu potřebnou pro reload po chybě aplikace v závislosti na výchozí aplikaci, dobu potřebnou pro restart systému včetně načtení výchozí aplikace. Všechny tyto parametry jsou prvně ověřeny na systému bez doinstalovaných aplikací (systém nese jen předinstalované aplikace od Microsoftu).

Dalším netradičním sledovaným parametrem je spolehlivost vzdáleného debuggu projektu Visual Studio 2015, tedy jsou sledovány problémy s konektivitou a spolehlivost aplikace běžící v debugg modu.

3.2. Metodika testování

3.2.1 Test aplikace IoT scénářů

V tomto testu se na vzorových scénářích zkouší reálné nasazení systému v IoT scénářích. První scénář je velmi jednoduchou modelovou aplikací využívající I²C sběrnici a senzor BMP180 k vytvoření naprosto jednoduchého teploměru, který zobrazuje teplotu a tlak v místě umístění senzoru, může to být jak kancelář (či jiný vnitřní prostor), tak venkovní prostředí. Vzorových programů, které řeší tento hojný problém, je celá řada na různé platformy, což mě vedlo k zařazení tohoto scénáře do testování.

Druhá modelová aplikace simuluje informační stánek (bez dotykového ovládání) OC Chodov Praha. Vychází z webového prohlížeče použitého pro testování práce s internetem a je založena na použití online dostupné mapy centra, tedy pro svůj běh užívá webovou adresu obchodního centra. Rozdíl je v tom, že upravená aplikace neobsahuje adresní řádek, jen nabídku domů (tedy zobraz mapu). Aplikace webového prohlížeče je zde reálně zjednodušena na „přehrávač“ webových aplikací. (zdrojový kód je v příloze E)

V tomto testu se nehodnotí ani tak vytížení systému, které je zde hodně závislé na kvalitě aplikace, ale hodnotí se výsledné vyřešení scénáře, tedy to, jak si systém dokáže poradit i s méně kvalitní aplikací a tím pádem dokáže poskytnout požadované výstupy i méně zkušeným

programátorům. Jde zde především o zkoušku toho, jak rychle dokáže systém komunikovat s aplikací a jak kvalitně aproximuje realtime zpracování problému.

3.2.2. Test práce s internetem

Vyhodnocuje se vytížení HW a rychlost prohlížení internetu, dále pak kvalita přehrávání online videí a případná omezení používání online služeb. Naměřené parametry jsou srovnány s parametry získanými na desktopovém Windows 10 (64 bitové architektury) pomocí shodného zdrojového kódu (stejně aplikace). Shodná aplikace zaručuje, že nebude měření zatíženo chybou způsobenou rozdílnou optimalizací kódu. Cílem testu je zjistit, zda je možné kvalitně, i na Windows 10 IoT Core, provozovat matematické a kancelářské nástroje běžící v cloudu, webové aplikace a podobně.

Čas potřebný pro načtení webové služby je poměrně dlouhý, protože aplikace nemá žádnou optimalizaci kódu (jde o nekvalitní aplikaci), proto jsem měřil čas pomocí ručních stopek. S ohledem na užitou aplikaci bych se měl pohybovat v nejméně příznivých časech a podmínkách pro práci samotnou. (zdrojový kód je v příloze F)

Testování obsahuje práce se službami Microsoft Office 365 a Office Online, kdy pro ukládání dat je využito uložiště Onedrive. Dále je otestována služba online výpočtů Wolframalpha.com. Do testování je zahrnuta i velmi populární služba youtube.com a práce s vyhledávači.

3.2.3. Test reakční doby na akci GPIO

Test má zjistit, jaká je průměrná odezva na akci provedenou zařízením připojeným přes GPIO s využitím nejjednodušší možné aplikace. Modifikoval jsem aplikaci HW Hallow Word tak, že LED svítí při stisku tlačítka, což trvá jen, dokud je tlačítko stlačeno. Stisknutí tlačítka zapíše na pin logickou jedničku a úkolem aplikace je tento stav přerušením propsat na výstupní pin diody. Zpoždění dané vlastní aplikací je velmi malé, protože jde pouze o negaci hodnoty výstupního pinu při vyvolání přerušení, které je dáno změnou logického stavu na vstupním pinu. Pokud se na problém podívám jako elektrotechnik, celou aplikaci by šlo nahradit jedním kusem drátu, který by spojoval mikropsínač a diodu.

Po odzkoušení upravené aplikace pomocí vzdáleného ladícího programu jsem aplikaci nasadil pomocí Visual Studia. Po nasazení aplikace jsem do zapojení HW přidal sondy osciloskopu, jeden kanál na tlačítko a druhý kanál na diodu. Na osciloskopu jsem si zobrazil oba použité kanály a aktivoval si časové kurzory.

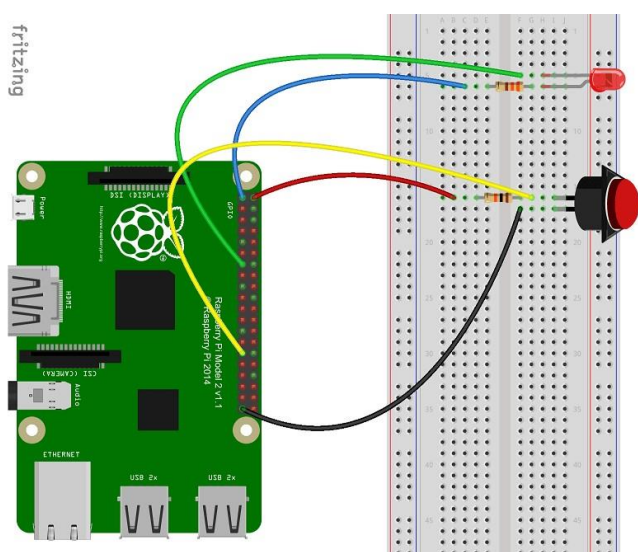
Test jsem započal spuštěním aplikace a vyčkáním na vypsání potvrzení o úspěšné inicializaci GPIO. Následně jsem si zkontroloval osciloskop a stiskl tlačítko. Po rozsvícení diody jsem okamžitě zamrazil obraz na osciloskopu. Nyní jsem na základě zobrazení obou kanálů pomocí časových kurzorů odečetl čas, který odděloval obě nástupné hrany u paty. Po odečtení jsem si čas zapsal do tabulky. Následně jsem osciloskop znovu spustil a měření zopakoval. Po 10 opakováních jsem restartoval Windows z důvodu, abych zamezil nějakému mimořádnému SW problému zkreslit výsledky. Následně jsem provedl dalších 10 měření.

Z naměřených hodnot jsem následně udělal průměr a určil chybu měření. Chybu měření jsem, po konzultaci s panem Ing. Martinem Šimůnkem z katedry měření elektrotechnické fakulty ČVUT, určil pomocí vztahu:

$$\delta = \frac{0,2}{\text{sample rate}} + \text{stálost_časové_základny [ppm]} \cdot \text{naměřená_hodnota} \quad (1)$$

Kde 0,2 představuje jitter vzorkování. Všechny parametry jsou dosazeny z dokumentace použitého osciloskopu.

Zdrojové kódy aplikace rozdělené dle jednotlivých souborů jsou umístěny v příloze C. Stejnou aplikaci v jazyce Python jsem připravil i pro Raspbian a provedl test úplně stejným způsobem, což mi umožnilo získat data pro srovnání jednotlivých systémů.



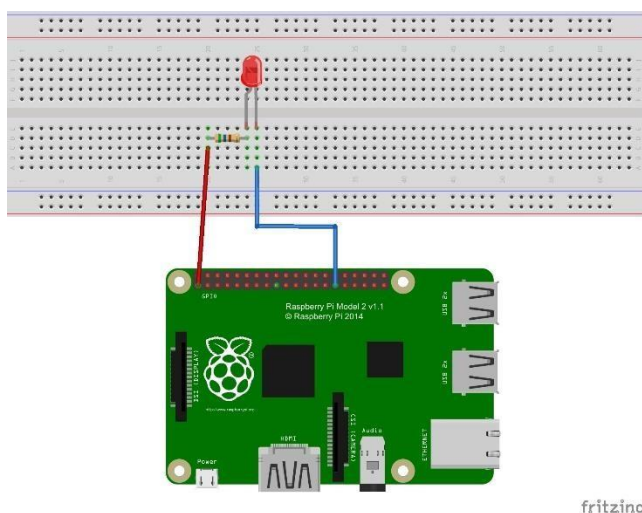
Obr. 42: Rozvržení HW (zdroj.: Chris Briggs: *Beginners guide to GPIO in Windows 10 IoT Core* [online]. 2015-07-06 [cit. 2015-11-22]. Dostupné z: <http://blog.chrisbriggssy.com/Beginners-guide-to-GPIO-Win10IoT-Core-Insider-Preview/>)

3.2.4. Test přesnosti SW časovačů

Díky nedostatku HW zdrojů hodinových signálů musí být často na většině zařízení typu Raspberry Pi2, která podporují Windows 10 IoT Core, doplněny tyto vývody o pin GPIO, který je naprogramovaný jako zdroj hodinového signálu. O řízení tohoto pinu se starají SW časovače, které jsou součástí aplikace. Nejednoduší aplikací, jejímž výkonným jádrem je časovač je, Blinky neboli HW Hallo World. Na této aplikaci, právě pro její malé nároky, jsem se rozhodl otestovat, jak přesný je hodinový signál, který řídí LED.

Celý test je opět jednoduchý, provedl jsem 4 sady měření po 5 hodnotách. K napájecímu kontaktu LED jsem připojil kanál osciloskopu. Nyní jsem (protože jde o symetrický signál) sledoval délku pulsu a následně délku rozestupu mezi pulsy v časové ose. V každé sadě jsem měřil jen délku, nebo rozestup, tedy celkově jsem změřil dva páry hodnot. Následně za každou sadu jsem spočítal průměrnou hodnotu a průměrnou odchylku od nastavené hodnoty. Ke každé hodnotě jsem stanovil příslušnou chybu měření. Všechny měřené hodnoty jsem postupně zapisoval do tabulky, kde jsem již měl předem připraveny hodnoty, které budou nasazeny do časovače. Aby aplikace nebyla bržděna uživatelským vstupem, který by nastavoval časovač, vygeneroval jsem několik verzí aplikace, které se od sebe liší právě nastavením časovače.

Výsledkem je odchylka mezi nastavenou hodnotou a hodnotou poskytovanou pinem v jednotlivých konfiguracích a také trend této odchylky. Chybu měření jsem, po konzultaci s panem Ing. Martinem Šimůnkem z katedry měření elektrotechnické fakulty ČVUT, určil dle vztahu (1). (zdrojový kód je v příloze D)



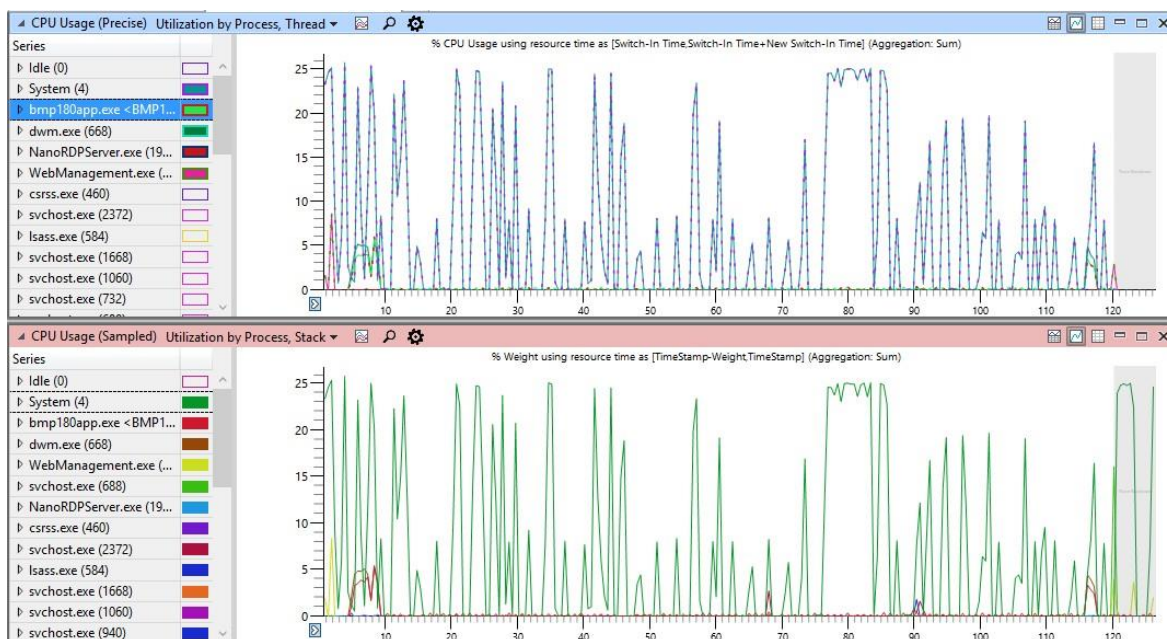
Obr. 43: Rozvržení HW (zdroj: Chris Briggs: Blinky Sample [online]. [cit. 2015-11-22]. Dostupné z: <https://ms-iot.github.io/content/en-US/win10/samples/Blinky.htm>)

3.3. Výsledky testů

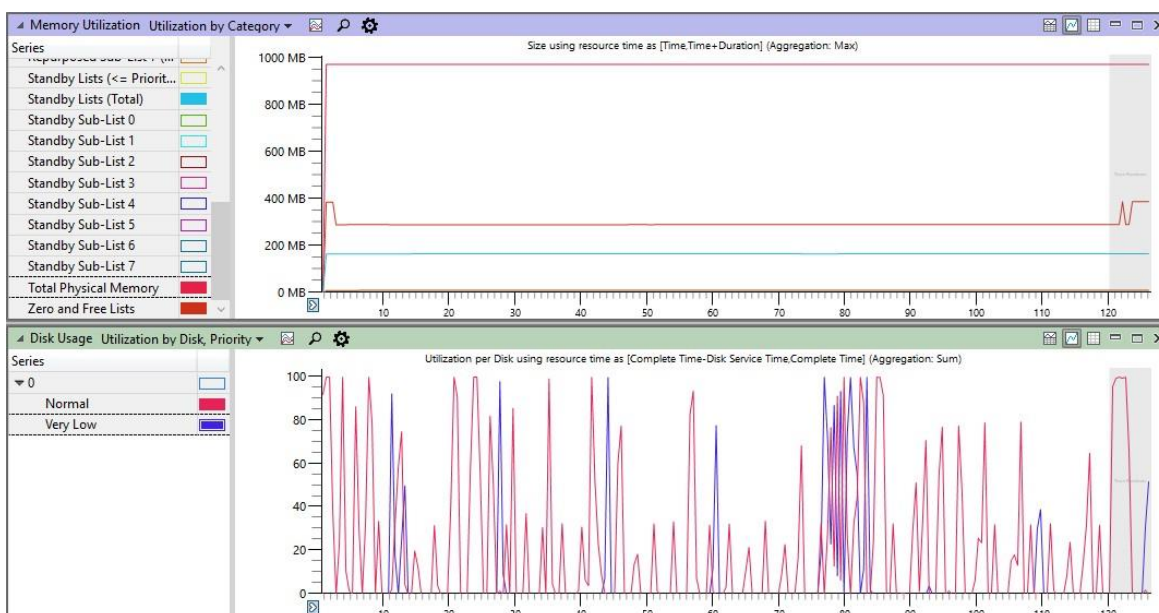
3.3.1. Test aplikace IoT scénářů

3.3.1.1. Teploměr

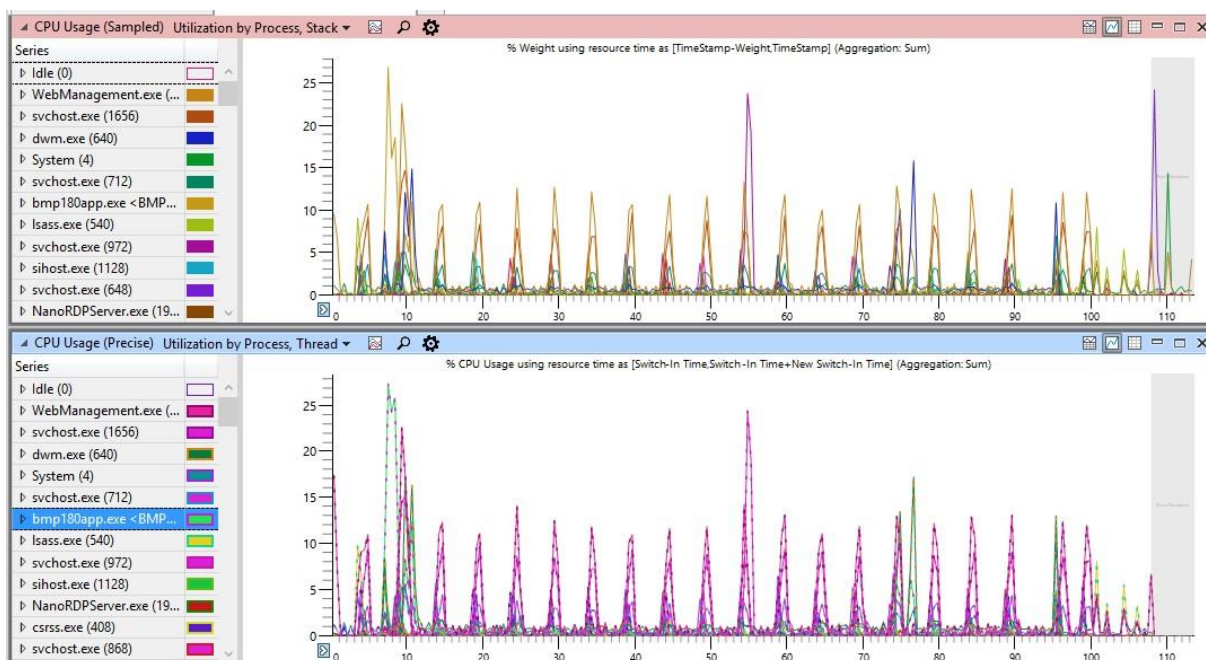
Aplikace je založena na senzoru BMP180. Aplikace běží stabilně a zcela plynule. Neprojevují se žádné problémy s pomalými reakcemi, na zahřívání i ochlazování senzoru, všechny reakce na změnu teploty jsou zcela plynulé.



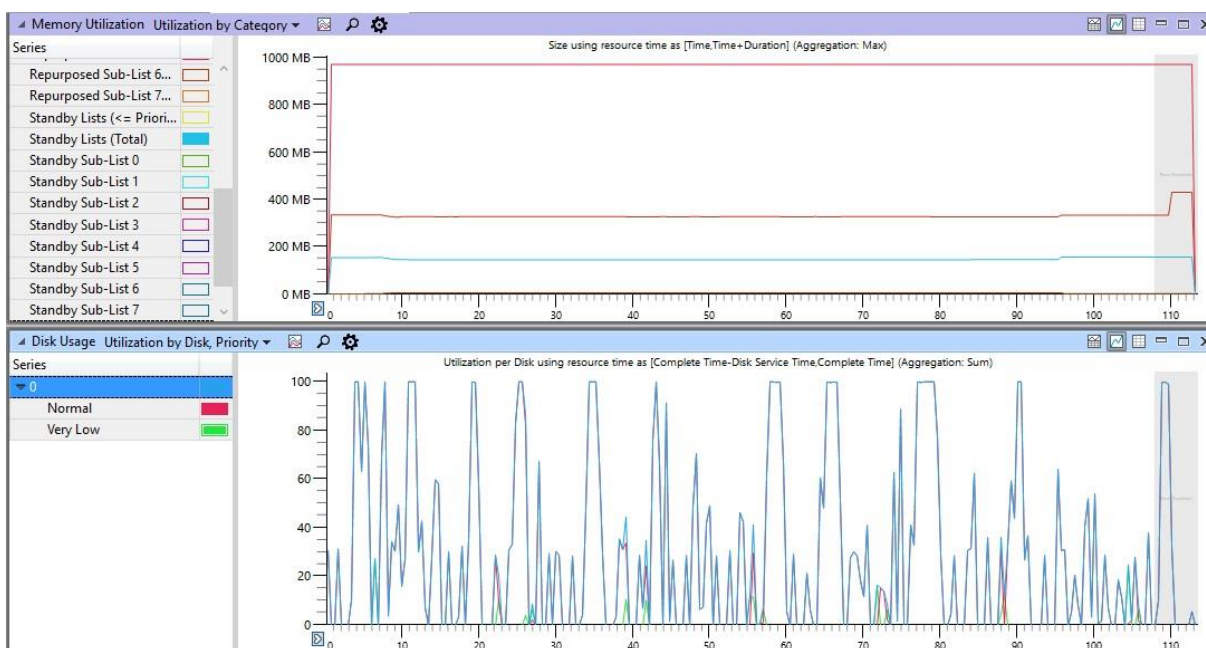
Obr. 42: CPU trace z běhu aplikace teploměr na Pi3



Obr. 43: Memory a Disk trace z běhu aplikace Teploměr na Pi3



Obr. 44: CPU trace z běhu aplikace Teploměr na Pi2



Obr. 45: Memory a Disk trace z běhu aplikace Teploměr na Pi2

Pokud se na provedení úlohy podíváme z pohledu spotřeby systémových prostředků, najdeme drobné rozdíly mezi jednotlivými deskami. Pi2 má mírně větší spotřebu fyzické paměti RAM, ale pokud jde o stránkovanou pohotovostní paměť, jsou na tom obě zařízení shodně. Dalších rozdílů si můžeme všimnout při pohledu na práci s diskem (paměťovou kartou), kde je vidět, že u Pi3 to potřebuje ještě hodně vylepšit, protože zde máme výrazně více přístupů k jednotce označených jako velmi pomalé, a to navzdory tomu, že obě desky disponovaly pro

běh aplikace kartou Kingston standartu U1. Tento rozdíl může mít vliv na plynulost běhu některých řešení, která hodně čtou či ukládají data právě na paměťovou kartu.

Pokud se podíváme na CPU stack, je vidět již znatelných rozdílů mezi deskami. Pi2 má výrazně větší spotřebu paměti zásobníku pro běh řídicího webserveru oproti Pi3, kde jednoznačně dominují základní systémové procesy. Spotřeba procesorového zásobníku přímo aplikací je naprosto minimální. Pokud se podíváme na spotřebu výpočetního času procesoru, pak na Pi2 po většinu sledované doby dominuje zpracování webserveru, zatím co na Pi3 je již jednoznačně dominantní běžící aplikace.

Na výše zmíněných rozdílech je vidět, že ačkoliv jsou systémy shodného buildu a navenek se aplikace chová shodně na obou deskách, jsou mezi systémy Windows 10 IoT Core pro Pi2 a Pi3 určité niance, o kterých si troufám říct, že nejsou dány jen rozdílnou architekturou obou desek, a tudíž jinými ovladači. Pozorování chování CPU stáčku je ve shodě s pozorováním chování při práci s webovým prohlížečem. Aplikace, které jsou náročné hodně využívají stack CPU poběží stabilněji na Pi3, protože jim zde systém nechává více prostoru, na druhou stranu pohled na využití výpočetního času ani serializace zpracování jednotlivých úloh na CPU (která je velmi zdařilá u obou desek a je shodná) nevysvětluje rozdíly v rychlosti běhu některých aplikací, které mluví ve prospěch slabší Pi2, nicméně na této konkrétní úloze pracující s I²C sběrnici to nebylo znatelné.

3.3.1.2. Infostánek

Aplikace informačního stánku, která je upravenou verzí webového prohlížeče, využívá tedy on-line aplikaci, která je dostupná na webu. Na Pi2 byla aplikace funkční, ale velmi pomalá a troufám si říct, že pro zákazníka neatraktivní, navzdory tomu, že byla stabilní. Nevyvolala ani zvýšené využití systémových prostředků ze strany OS, ale bylo znatelně cítit, že jde o kód, který není optimalizovaný a sám o sobě má velkou spotřebu systémových prostředků.

Naproti tomu, pokud aplikace běžela na Pi3, vyvolávala výrazně zvýšenou spotřebu prostředků systémem, pro což nemám žádné logické vysvětlení. Nešlo ani o jednorázový jev, šlo jen o opakovaný, který se projevil i po reinstalaci systému. Z pohledu uživatele byla aplikace stabilní a poměrně živá, zde bych si dokázal představit i její reálné použití.

3.3.2. Test práce s internetem

Po pravidelném používání aplikace na Pi2 i desktopu musím konstatovat, že pády aplikace, ke kterým občas na Pi2 došlo, provází i desktopovou verzi programu. Jediný problém, na který jsem narazil ve verzi pro Pi2 bylo, že někdy dojde k pádu aplikace při práci s OneDrive z outlook.com a parametrizovaném vyhledávání na Youtube, což se na Pi3 neděje a na desktopu také ne. Pro všechny 3 verze aplikace platí, že Office online je dostupný pouze v režimu pro čtení dokumentu, nikoliv pro jeho editaci či tvorbu. Toto omezení Office online souvisí s konkrétním užitým zdrojovým kódem, nikoliv s OS, na kterém běží.

Pokud jde o procházení internetu, jsem Pi2 používal především jako hudební a video přehrávač Youtube a následně pro čtení článků. S Pi3 jsem pracoval výrazně kratší dobu, navzdory opravdu markantnímu rozdílu v rychlosti prohlížení webu je aplikace na Pi3 zřejmě stabilnější, hodně je to znát například na PowerBI portálu, kde se velmi spolehlivě dají dělat editace a rychleji aplikuje dynamické filtry proti Pi2.

Srovnání průměrných časů, za které se na jednotlivých platformách načítaly vybrané webové stránky (bulvár je tam z důvodu náročnosti stránky), je vidět v následující tabulce.

Stránka / služba	Čas načtení stránky či služby [s]		
	Pi2	Pi3	X64
ttp://jcu.cz	2,6 ± 0,2	6,7 ± 0,2	1,8 ± 0,2
http://google.cz	1,1 ± 0,2	1,3 ± 0,2	1,0 ± 0,2
Vyhledání pojmu „IoT“ pomocí Google	1,8 ± 0,2	4,0 ± 0,2	1,6 ± 0,2
Seznam.cz	11,5 ± 0,2	31,1 ± 0,2	6,5 ± 0,2
http://youtube.com	8,1 ± 0,2	22,5 ± 0,2	8,7 ± 0,2
Spuštění přehrávání vybraného videa na youtube	14,9 ± 0,2	31,8 ± 0,2	4,5 ± 0,2
Stram.cz	10,9 ± 0,2	11,7 ± 0,2	3,2 ± 0,2
Načtení epizody Mistrů volantů na Stream.cz	14,7 ± 0,2	15,6 ± 0,2	2,0 ± 0,2
Office portál	3,2 ± 0,2	6,4 ± 0,2	2,0 ± 0,2
Word online (otevření 22 stran dokument ve formátu docx)	15,0 ± 0,2	16,6 ± 0,2	4,9 ± 0,2
PowerPoint online (otevření prezentace 18 snímků)	14,3 ± 0,2	11,5 ± 0,2	7,2 ± 0,2
SharePoint online (JCU)	17,4 ± 0,2	21,2 ± 0,2	
OneDrive	33,8 ± 0,2	43,0 ± 0,2	9,6 ± 0,2
https://www.wolframalpha.com/	(2,2 / 12,2*) ± 0,2	(6,7 / 14,2*) ± 0,2	(- / 6,4*) ± 0,2
Výpočet výrazu: $\sin(\sqrt{X^2+Y^2})/(\sqrt{X^2+Y^2})$ pomocí služby Wolfram Alpha	20,8 ± 0,2	24,8 ± 0,2	8,8 ± 0,2
PowerBI portál	(6,1 / 39,6**) ± 0,2	(2,6 / 53,3**) ± 0,2	1,6 ± 0,2 / -
Živě.cz	19,3 ± 0,2	15,6 ± 0,2	7,9 ± 0,2
Blesk.cz	17,4 ± 0,2	12,9 ± 0,2	10,9 ± 0,2

Tab. 4: Časy načítání webových služeb a stránek

Časy v tabulce označené * jsou zapsány ve formátu *mobilní / plná verze webové stránky* a časy označené ** jsou zapsány ve formátu *načtení webu / přihlášení a zobrazení Dashboardu*. Časy uvedené v tabulce jsou průměrem za více pokusů o načtení webové stránky či služby. Chyba měření uvedená v tabulce je standardní chyba pro použití ručních stopek a souvisí s lidskou reakční dobou.

Jako platforma X64 byl použit Windows Server 2016 TP5 na zařízení Toshiba Satellite L750 1L8 ve výrobní konfiguraci. Důvodem bylo, že na tomto systému byl prohlížeč používán nejčastěji, protože na rozdíl od výchozího IE11 dokázal přehrát youtube a stream. Nevýhodou takto primitivní aplikace je, že dochází k automatickému přesměrování určitých odkazů do výchozího prohlížeče, a to platí především pro služby outlook.com (nelze přejít z pošty na OneDrive), PowerBI a SharePoint.

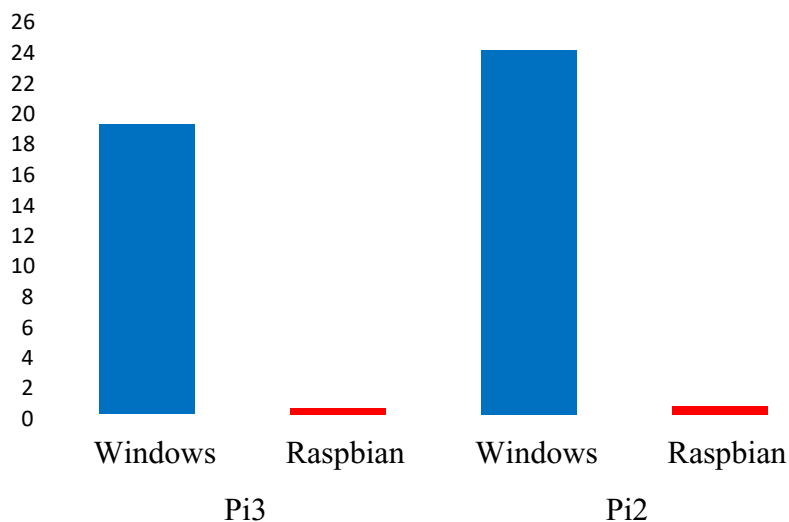
3.3.3. Test reakční doby na akci GPIO

Číslo měření	Windows 10 IoT Core		Raspbian	
	Odezva [ms] Pi3	Odezva [ms] Pi2	Odezva [μs] Pi3	Odezva [μs] Pi2
1	8,0 ± 0,2	9,0 ± 0,3	520 ± 13	540 ± 13
2	5,2 ± 0,1	10,4 ± 0,3	420 ± 11	560 ± 14
3	26,0 ± 0,7	5,8 ± 0,2	410 ± 10	500 ± 13
4	7,0 ± 0,2	13,6 ± 0,3	410 ± 10	470 ± 12
5	20,0 ± 0,5	47,0 ± 1,2	400 ± 10	540 ± 14
6	42,0 ± 1,0	7,4 ± 0,2	380 ± 90	550 ± 14
7	18,0 ± 0,5	18,0 ± 0,5	420 ± 10	590 ± 15
8	1,0 ± 0,02	22,8 ± 0,6	420 ± 10	500 ± 13
9	4,6 ± 0,1	17,8 ± 0,5	420 ± 10	560 ± 14
10	24,0 ± 0,6	42,0 ± 1,0	410 ± 10	620 ± 16
11	14,8 ± 0,4	30,0 ± 0,8	420 ± 10	510 ± 13
12	19,6 ± 0,5	31,6 ± 0,8	420 ± 10	510 ± 13
13	38,0 ± 0,5	27,2 ± 0,7	410 ± 10	400 ± 10
14	9,0 ± 0,2	44,0 ± 1,1	410 ± 10	510 ± 13
15	55,6 ± 1,3	29,6 ± 0,7	430 ± 11	510 ± 13
16	19,6 ± 0,5	22,8 ± 0,6	410 ± 10	430 ± 10
17	12,2 ± 0,3	9,8 ± 0,3	420 ± 11	530 ± 13
18	28,0 ± 0,7	24,4 ± 0,6	410 ± 10	660 ± 17
19	11,4 ± 0,3	26,0 ± 0,7	410 ± 10	520 ± 13
20	16,4 ± 0,4	38,0 ± 0,9	410 ± 10	500 ± 13
Průměrná odezva	19,0 ± 0,5	23,9 ± 0,6	418 ± 11	526 ± 13

Tab. 5: Naměřené časy odezvy

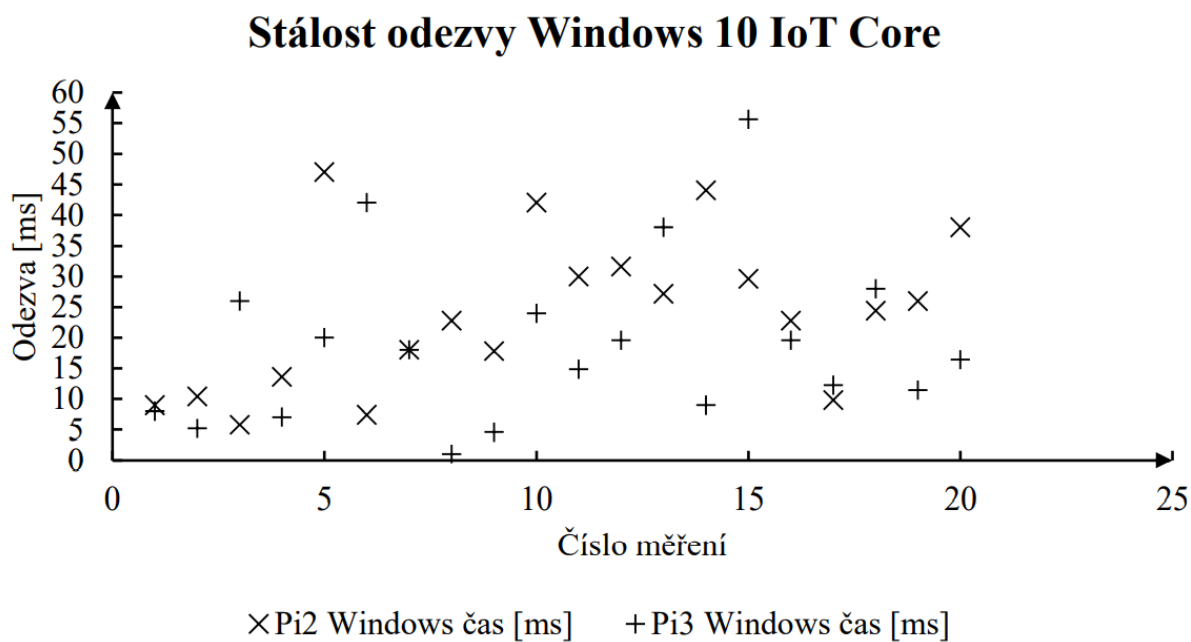
Z výše uvedené tabulky je jednoznačně patrné, že Raspbian má řádově lepší odezvu nežli Windows. Ale na druhou stranu musím být trochu objektivní, signál, který jsem dostal z Raspbianu, měl mnohem více překmitů a celkově nižší kvalitu ve srovnání s Windows, které vracely učebnicový signál. Bohužel na fotografiích to není příliš patrné z důvodu, že zákmity překryl časový kurzor. Ve Windows aplikace běžela v grafickém režimu, jak je požadované pro Universall Apps, naproti tomu Raspbian celý běžel bez grafické nadstavby, což v kombinaci s kvalitním interpretem Pythonu přispělo k velmi rychlým reakcím systému. Vzhledem k rychlému rozvoji Windows 10 IoT Core můžeme předpokládat do budoucna zlepšení. Mnohem lepší pohled na získaná data nám dává následující graf, který shrnuje průměrné výsledky ze všech 4 měření a umožňuje tak velmi přehledné porovnání výsledků.

Průměrná odezva GPIO sběrnice [ms]

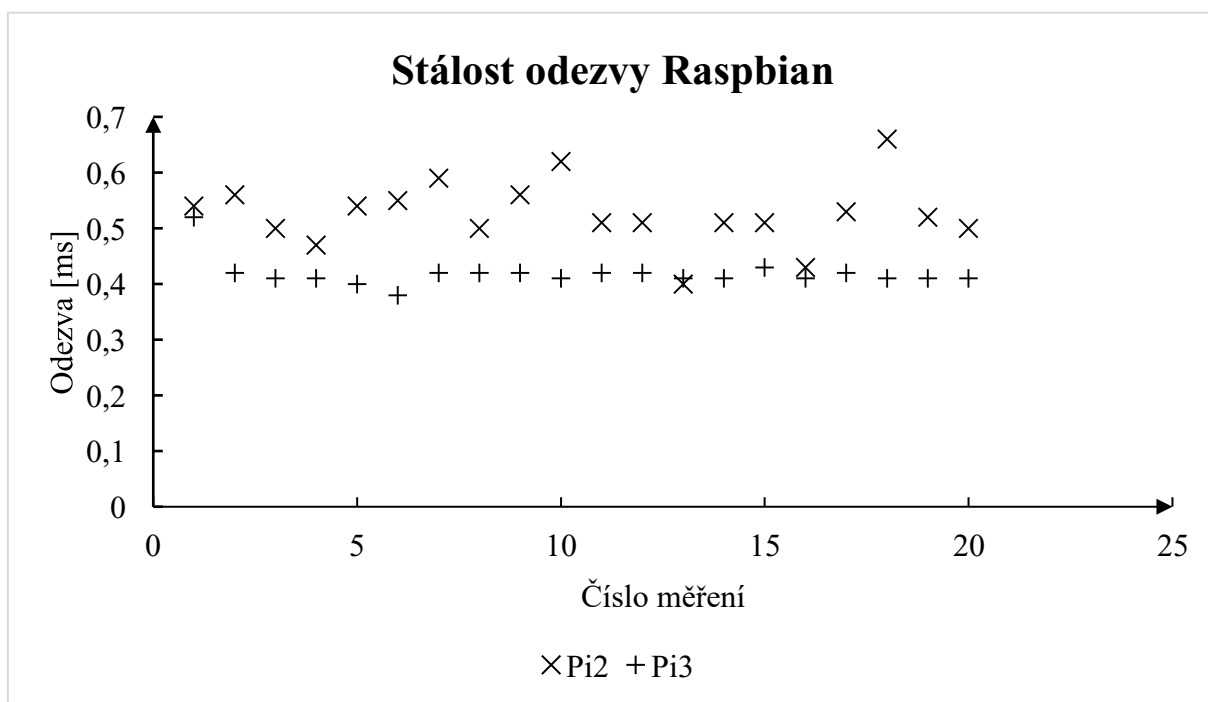


Graf č. 1: Srovnání průměrné rychlosti odezvy Windows a Raspbianu na Pi2 a Pi3

Dalším faktorem, který mluví proti nasazení Windows tam, kde je potřeba rychlá odezva na GPIO akci, je nestálost odezvy, která je výrazně větší, nežli je tomu u Raspbianu. Důvod této nestálosti odezvy neznám, protože jsem nerozebíral Windows 10 SDK, respektive Windows IoT Extensions for the UWP. Nejlépe je srovnání obou systémů patrné opět na grafech, které ovšem tentokrát nelze (především v případě Windows) proložit nějakou křivkou. Očekávanému průběhu (rovnoběžka s osou x) se nejlépe blíží Raspbian na Pi3.



Graf č. 2: Stálost odezvy Windows 10 IoT Core



Graf č. 3: Stabilita odezvy Raspbian

3.3.4. Test přesnosti SW časovačů

Číslo měření	Naměřená hodnota [ms]	Nastavená hodnota [ms]	Rozdíl [ms]
1	$29,4 \pm 7,35 \cdot 10^{-1}$	0,1	29,3
2	$13,4 \pm 3,35 \cdot 10^{-1}$	0,1	13,3
3	$18,2 \pm 4,55 \cdot 10^{-1}$	0,1	18,1
4	$16,0 \pm 4,00 \cdot 10^{-1}$	0,1	15,9
5	$14,6 \pm 3,65 \cdot 10^{-1}$	0,1	14,5
Průměr	$18,3 \pm 4,58 \cdot 10^{-1}$	0,1	18,2
6	$31,6 \pm 7,90 \cdot 10^{-1}$	1	30,6
7	$37,6 \pm 9,40 \cdot 10^{-1}$	1	36,6
8	$37,2 \pm 9,30 \cdot 10^{-1}$	1	36,2
9	$31,2 \pm 7,80 \cdot 10^{-1}$	1	30,2
10	$31,6 \pm 7,90 \cdot 10^{-1}$	1	30,6
Průměr	$33,8 \pm 8,45 \cdot 10^{-1}$	1	32,8
11	$33,4 \pm 8,35 \cdot 10^{-1}$	10	23,4
12	$31,4 \pm 7,85 \cdot 10^{-1}$	10	21,4
13	$31,8 \pm 7,95 \cdot 10^{-1}$	10	21,8
14	$25,2 \pm 6,30 \cdot 10^{-1}$	10	15,2
15	$31,4 \pm 7,85 \cdot 10^{-1}$	10	21,4
Průměr	$30,6 \pm 7,65 \cdot 10^{-1}$	10	20,6
16	$536,0 \pm 1,34 \cdot 10$	500	36,0
17	$508,0 \pm 1,27 \cdot 10$	500	8,0
18	$516,0 \pm 1,29 \cdot 10$	500	16,0
19	$528,0 \pm 1,32 \cdot 10$	500	28,0
20	$532,0 \pm 1,33 \cdot 10$	500	32,0
Průměr	$524,0 \pm 1,31 \cdot 10$	500	24,0
21	$784,0 \pm 1,96 \cdot 10$	750	34,0
22	$768,0 \pm 1,92 \cdot 10$	750	18,0
23	$788,0 \pm 1,97 \cdot 10$	750	38,0
24	$780,0 \pm 1,95 \cdot 10$	750	30,0
25	$780,0 \pm 1,95 \cdot 10$	750	30,0
Průměr	$780,0 \pm 1,95 \cdot 10$	750	30,0

Tab. 6: Naměřené hodnoty přesnosti časovače Pi2

Číslo měření	Naměřená hodnota [ms]	Nastavená hodnota [ms]	Rozdíl [ms]
1	$15,2 \pm 3,80 \cdot 10^{-1}$	0,1	15,1
2	$16,0 \pm 4,00 \cdot 10^{-1}$	0,1	15,9
3	$15,6 \pm 3,90 \cdot 10^{-1}$	0,1	15,5
4	$15,4 \pm 3,85 \cdot 10^{-1}$	0,1	15,3
5	$15,4 \pm 3,85 \cdot 10^{-1}$	0,1	15,3
Průměr	$15,5 \pm 3,88 \cdot 10^{-1}$	0,1	15,4
6	$34,0 \pm 8,50 \cdot 10^{-1}$	1	33,0
7	$32,0 \pm 8,00 \cdot 10^{-1}$	1	31,0
8	$28,6 \pm 7,15 \cdot 10^{-1}$	1	27,6
9	$27,6 \pm 6,90 \cdot 10^{-1}$	1	26,6
10	$31,0 \pm 7,75 \cdot 10^{-1}$	1	30,0
Průměr	$30,6 \pm 7,65 \cdot 10^{-1}$	1	29,6
11	$35,6 \pm 8,90 \cdot 10^{-1}$	10	25,6
12	$31,6 \pm 7,90 \cdot 10^{-1}$	10	21,6
13	$31,8 \pm 7,95 \cdot 10^{-1}$	10	21,8
14	$27,0 \pm 6,75 \cdot 10^{-1}$	10	17,0
15	$27,4 \pm 6,85 \cdot 10^{-1}$	10	17,4
Průměr	$30,7 \pm 7,68 \cdot 10^{-1}$	10	20,7
16	$524,0 \pm 1,31 \cdot 10$	500	24,0
17	$512,0 \pm 1,28 \cdot 10$	500	12,0
18	$518,0 \pm 1,30 \cdot 10$	500	18,0
19	$518,0 \pm 1,30 \cdot 10$	500	18,0
20	$536,0 \pm 1,34 \cdot 10$	500	36,0
Průměr	$521,6 \pm 1,30 \cdot 10$	500	21,6
21	$788,0 \pm 1,97 \cdot 10$	750	38,0
22	$780,0 \pm 1,95 \cdot 10$	750	30,0
23	$772,0 \pm 1,93 \cdot 10$	750	22,0
24	$772,0 \pm 1,93 \cdot 10$	750	22,0
25	$780,0 \pm 1,95 \cdot 10$	750	30,0
Průměr	$778,4 \pm 1,95 \cdot 10$	750	28,4

Tab. 7: Naměřené hodnoty přesnosti časovače Pi3

Jak je patrné z hodnot v tabulkách a také z fotografií znázorňujících průběh výstupního signálu, Pi3 poskytuje celkově kvalitnější výstup. I tak je na první pohled patrné, že časovač zdaleka nedosahuje přesnosti ani stability potřebné pro další řízení HW. Jeho použití je bez problémů možné pro vytvoření jednoduchých stopek, nebo řízení osvětlení, či LED „hada“ nebo pro tónové generátory ve slyšitelném spektru bez požadavků na přesnost tónu za účelem ladění hudebních nástrojů. Vzhledem k nedostatku HW časovačů na deskách Pi2 a Pi3 lze časovače pro řadu praktických úloh (např.: načítání dat ze senzorů v řádově minutových intervalech) zcela bez obav použít.

4. Ukončení podpory

Systém Windows 10 přišel 29.06.2015 jako velmi nadějný multiplatformní operační systém. Do konce roku 2016 se na trhu objevily prakticky všechny jeho edice. Jako poslední na trh mezi edicemi vstupují edice Pro Education a Pro for Workstation. Systém zvládl pokrýt všechny platformy, od mobilních telefonů (Windows 10 Mobile) přes počítače, tablety (Windows 10, Windows 10 S a Windows 10 on ARM), servery (Windows server 2016 a 2019) až po IoT (Windows 10 IoT Core a Windows 10 IoT Enterprise). Kategorii samu pro sebe je Windows 10 Team, který speciální edicí pro zařízení Surface Hub.

S postupem času se začalo ukazovat, že multiplatformní řešení na úrovni Windows není pro Microsoft nejspíš ideálním. První velkou platformou, kterou Windows opouští jsou mobilní telefony, Windows 10, na rozdíl od svého předchůdce, jsou dostupné jen v telefonech Microsoftu a firma se rozhodla ukončit jejich podporu s koncem roku 2019.

Další oblastí zařízení, ze kterých se Windows stahují jsou IoT zařízení pro širokou veřejnost a školství, kam patří právě Raspberry Pi. Z obchodního hlediska cílila edice IoT Core na kutily, kde spoléhala na propojení se službami Azure, což mělo financovat podporu a vývoje systému. Druhou obchodní kategorií, kam edice IoT Core cílila bylo školství, kde měla zprostředkovat seznámení s IoT a mezipředmětové vztahy Informatika – Fyzika. Cílem bylo, aby se žáci učili programovat UWP a své programátorské úsilí z hodin informatiky přenesli do hodin fyziky. Přímou Microsoft vytvářel pro fyzikáře řadu již hotových experimentů, které využívali Windows 10 IoT Core na Raspberry Pi a služeb Azure pro zpracování dat. Tento model se ukázal jako milný z obchodního hlediska, a proto dle zásad moderního životního cyklu SW společnosti Microsoft byla podpora poslední verze Windows 10 IoT Core ukončena 12.11.2019. Poslední vydanou verzí tohoto systému je verze 1803 jak ve stabilní, tak v Insiders verzi. Insiders verze podporuje Raspberry Pi3B+, zatím co stabilní verze podporuje pouze Raspberry Pi3.

Ústup z IoT světa se netýká firemního sektoru, kde Microsoft nadále dodává nástupce Windows Embedded a Industry, což je řada Windows 10 IoT Enterprise. Nově se v tomto sektoru objevují také serverové verze, které jsou určeny pro běh na specifických zařízeních.

Navzdory oficiálním prohlášením zástupců společnosti Microsoft z let 2015 až 2019 nejsou Windows 10 poslední generací systému Windows. Všechny verze Windows vyjímaje LTSC edice verzí budou mít ukončenou podporu 14.10.2025. Dne 5.10.2021 vyšly Windows 11, které jsou následníkem Windows 10 a pro majitele desítek jsou volitelným bezplatným upgradem.

Podpora Windows 10 jako celku (při pravidelných aktualizacích verzí) je velkým paradoxem, systém, který měl být poslední generací tak odchází po 10 letech jako předešlé legendární i nenáviděné verze Windows. Málo lidí využívá a ví, že Windows 8.1, které širokou veřejností nebyly příliš úspěšně přijaty jsou v roce 2021 stále podporovaným systémem a jejich podpora skončí v roce 2023, tedy po 10 letech. Deseti let podpory se dočkala rovněž legenda v podobě Windows 7. Delší podpory se dostalo jen nejúspěšnějšímu systému Windows, tedy Windows XP, které se pyšnily podporou celých 13 let. Zde je dobré si uvědomit, že podporu v celkové délce 10 let mají zpravidla Windows Servery, nikoliv klientské Windows. U tohoto odstavce je nutné poznamenat dvě zásadní věci, tou první je, že se zde nebavíme o Embedded systémech, systémech v edicích LTCS (poslední zástupce longstable verze končí podporu až v roce 2029), tyto systémy se neřídí moderními zásadami životního cyklu SW. Tou druhou je fakt, že se zde nebavíme o smluvním prodloužení podpory pro konkrétní organizaci nebo státní správu, které si Microsoft nechá řádně zaplatit.

4.1. Odkaz Windows 10 IoT Core

Ač se může zdát Windows 10 IoT Core jako slepou vývojovou verzí, což v úzce zaměřeném pohledu je, využíváme z nich v běžných i serverových Windows celou řadu prvků. Nemálo vývojářů i správců využívá portál zařízení, který je dostupný v Nastavení => Pro vývojáře. Portál zařízení umožňuje plnohodnotné Windows ovládat vzdáleně přes webový prohlížeč stejně, jako tomu bylo u IoT Core, jen nabízí více možností. Tento portál je také dobrým nástrojem pro správce v malých prostředích, kteří přes něj mohou spravovat zařízení vývojářů.

Na stejném technickém základu jako webové prostředí pro správu Windows 10 IoT Core, tedy PowerShellu a websocketu, stojí také bezplatný nástroj pro správce menších sítí zvaný Windows Admin Center. Windows Admin Center je rovněž doporučenou grafickou cestou k ovládání Windows serveru, které jej umožňují od verze 2019 stáhnout jedním kliknutím z Windows server konzole. Admin center podporuje rovněž instalaci Core systému Windows server od verze 2012. Při současném trendu jde patrně o nástupce RSAT. Z osobní zkušenosti mohu doporučit Windows Admin center jako nástroj pro správce škol, provozují jej v síti s téměř 200 klientskými Windows zařízeními a 4 servery.

4.2. Co dál s projekty pro Windows 10 IoT Core?

Microsoft nadále na oficiálním webu umožňuje stažení posledního dostupného buildu Windows 10 IoT Core a nástrojů potřebných pro jeho nasazení i provoz (platí ke dni 20.10.2021). Pro správu i nasazení systému lze použít PC s Windows 10 i 11, zde nedošlo k omezení (vlastnoručně testováno). Byť to nelze z pohledu bezpečnosti doporučit, je možné stále na podporovaných deskách systém provozovat spolu s původními projekty. Navzdory ukončení podpory se pro systém brány Azure neuzavřely, přístup k Azure službám je čistě na aplikační úrovni.

Oficiálně podporovaný scénář ze strany Microsoftu pro další provoz projektů vzniklých pro Windows 10 IoT Core (zde nemyslím multimediální centra či kiosky) zde není. Pokud byl systém Windows IoT Core využit v rámci korporátního řešení, je možné s využitím Windows 10 IoT Enterprise pokračovat s projekty využívající I²C nebo GPIO.

Neoficiální cestou je projekt WoR (Windows on Raspberry), který dovolí rozběhnout desktopové Windows 10 Home a Pro či Windows 11 Home a Pro s dostupnou sběrnici GPIO na úrovni ovladačů (ovladač je přítomen). Výhodou tohoto scénáře je, že uživatel dostává bezpečnostní a další aktualizace od Microsoftu, nevýhodou jsou nepodepsané, komunitní ovladače, které nelze aktualizovat a chybějící podpora. Může se stát, že v případě Windows 11 bude uživatel odkázán na Windows update catalog. Velkou nevýhodou je pak výkon celého řešení, protože v systému musí běžet celá řada procesů, které Windows 10 IoT Core neměly.

4.2.1. Vzdal Microsoft multiplatformnost a domácí IoT?

Ač se to tak může jevit, nevzdal. .NET tým přišel na řadu s druhým pokusem, který se snad již vydaří lépe. Microsoft už delší dobu vyvíjí multiplatformní verzi .NET, která dokáže fungovat i na Linuxu a macOS. Zprvu tato verze byla označena jako Core, protože neumožňovala tolik věcí jako plnohodnotná verze pro Windows. Dnes (říjen 2021) je již dostupná verze .NET 5, která je multiplatformní a „plnohodnotná“ i pro Windows. Právě .NET 5 je tím druhým pokusem Microsoftu, protože je oficiálně podporován na Raspberry Pi OS, což umožňuje programovat multiplatformní i IoT aplikace. Tyto IoT aplikace pak dokážou využívat služeb MS Azure a do značné míry zjednodušují programování IoT aplikací na platformě Microsoft. Microsoft díky tomu ušetří na vývoji jedné verze systému Windows a naplní obchodní model s využitím toho, co je vyvíjeno primárně pro jiné účely.

5. Projekt WoR

Projekt Windows on Raspberry (WoR: <https://www.worproject.ml/>) není žádnou novinkou. Již od časů Pi2 a Windows 10 IoT Core se členové komunitního projektu snažili dostat plnohodnotné Windows na Raspberry Pi. Dnes musím uznat, že na Pi4 jde díky tomuto projektu provozovat s určitými řešitelnými limity Windows 10 i 11. Projekt se stará o firmware, a ovladače, obrazy systému jsou originální z Microsoftu, byť stažené neoficiálním nástrojem. (*Windows on Raspberry* [online]. [cit. 2021-10-26]. Dostupné z: <https://www.worproject.ml/>)

Pro účely této práce jsem stažené Windows 10 ARM upravil po vzoru image, které chystám pro svého zaměstnavatele, úprava je tedy mé Know-How, které nevyzradím, až na úpravu konfigurace napájení.

5.1. Nasazení Windows dle WoR

V první řadě je potřeba upozornit, že pokud je nastaveno DNS na 1.1.1.3, nebo 1.1.1.2, nelze na stránky projektu WoR přistoupit. Uvedená DNS patří službě 1.1.1.1 for Families a znamenají blokaci malware a obsahu pro dospělé, v druhém případě pouze blokaci malware. (*Introducing 1.1.1.1 for Families* [online]. San Francisco, 2020 [cit. 2021-10-20]. Dostupné z: <https://blog.cloudflare.com/introducing-1-1-1-1-for-families/>) Po stažení Image nástroje WoR_Release_2.2.2 a jeho vybalení ze zipu je možné spustit WoR.exe. Musím upozornit, že dle Virustotal obsahuje soubor dvojici nákaz (viz: [VirusTotal - File - 98c22763b2057fa1df87e36a29d0df17f5fad30d295f631978785a49ff91b80b](#)), na druhou stranu u velkých hráčů jako Kaspersky, Bitdefender, Eset a Microsoft je soubor veden jako čistý.

Problém nastává po spuštění WoR.exe, kdy vše funguje dobře, ale při opakovaném užití na stejném PC se mi několikrát stalo, že jeho proces byl v průběhu nasazování systému ukončen antimalwarovým řešením společnosti Kaspersky. Nástroj na pozadí volá nativní nástroj DSM.exe a k terminaci procesu dochází až v průběhu nasazení Windows, tedy ve fázi mezi příkazy DSM.exe /Apply-Image a závěrečným ověřením (nejspíše při instalaci ovladačů: Dism /Image:<Path> /Add-Driver /Driver:<Path> /Recurse). Zkoušel jsem nástroj před spuštěním ověřit řešením společnosti Kasperky a byl označen jako čistý, je jediným důvodem práce s dotaženými ovladači či firmware. Pokud tedy považuje uživatel nástroj za bezpečný, doporučím vypnout antivirovou ochranu třetích stran (úplné ukončení antimalwarového řešení) na dobu instalace.

5.1.1. Potřebný HW

Pro nasazení Windows 10 on ARM (případně Windows 11 on ARM) je potřeba externí uložení, které bude na Pi4 systémovým diskem. Doporučit mohu externí SSD jednotku připojenou přes USB 3.0, případně certifikovaný USB Flash disk pro technologii Windows to Go (viz [dokumentace](#)). Osobně jsem neudělal dobrou zkušenost s 2,5“ HDD WD Black 7200 RPM připojeným přes redukci SATAIII na USB 3.0, systém se hodně trhá z důvodu nedostatečného napájení disku. Pro rychlé 2,5“ HDD bych doporučil USB čtečku HDD s vlastním napájením ze sítě 230 V. Z microSD karet mohu doporučit 3D Memory Cards for IoT, což jsou karty určené pro průmyslové nasazení v IoT od renomovaných výrobců. S vědomím omezené životnosti je pro domácí využití dostatečná microSD karta standardu UHS-I/U3 V30, které poskytují dostatečně rychlé uložení pro relativně plynulý chod systému a z vlastní zkušenosti si troufám říct, že nejsou úzkým hrdlem provozu.

Vlastní RPi si vyžádá další příslušenství dle své verze, pro Pi4 potřebujeme redukci z microHDMI na HDMI, pro Pi3 stačí obyčejné HDMI. Osobně bych volil pro nasazení Windows on ARM Pi4, na Pi3 se více hodí Windows 10 IoT Core. Pro všechny verze potřebujeme Ethernetový kabel CAT5e nebo vyšší, současné ovladače nepodporují vestavěnou Wi-Fi. USB Wi-Fi karty jsem nezkoušel, proto nemohu soudit, zda a které jsou vhodné. Samozřejmostí jsou periferie jako je klávesnice, myš a monitor či jiný zobrazovač.

Posledním potřebným kusem HW je počítač s podporovanou verzí systému Windows. Osobně jsem celým procesem stažení a nasazení image Windows prošel na Windows 11 Insiders Preview, nasazením image Windows jsem také vyzkoušel na Windows 10 verze 21H1. S ohledem na využívané systémové nástroje si dovoluji tvrdit, že celý proces bude bez obtíží fungovat i na Windows 8.1 update 1.

Dle zkušeností členů komunity projektu WoR bych doporučil Raspberry Pi osadit dodatečným aktivním chlazením a přetaktovat jej na 2 GHz, případně 2,3 GHz, při patřičné konfiguraci chlazení v UEFI. Vlastní přetaktování je možné v UEFI nebo pomocí nástroje projektu WoR.

5.1.2. Potřebné úpravy konfigurace UFI

Při prvním spouštění systému je potřeba přejít klávesou ESC do konfigurace zařízení a lehce ji pozměnit. V sekci Device Manager => Raspberry Pi Configuration doporučuji přenastavit Display configuration na 1080p, nebo 720p, a vypnout Native Resolution. Dalším nutným krokem je na Pi4 vypnutí limitu RAM, to se provede v sekci Device Manager => Raspberry Pi Configuration => Advanced Configuration. Doporučuji v sekci Boot Maintenance Manager změnit Boot order tak, aby na prvním místě byla jednotka s Windows. Upozorňuji, že pokud bude střídáno více uložišť s různými verzemi nebo konfiguracemi Windows, je potřeba tato nastavení provést po každé, když je jednotka připojena prvně.

5.1.3. Doporučená úprava konfigurace Windows po prvním spuštění

Úplně prvním krokem na běžných PC je aktivace systému Windows, u Raspberry Pi doporučuji tomuto kroku předradit dvojici změn. Prvním krokem je aktivovat napájecí schéma „Špičkový výkon“:

1. Spustit cmd jako správce
2. Příkaz: `powercfg -duplicatescheme 46ef2308-0c99-4d91-860b-bb3ff61bc457`
 - a. Windows 11: `powercfg -duplicatescheme e9a42b02-d5df-448d-aa00-03f14749eb61`
3. Příkaz: `powercfg /setactive 46ef2308-0c99-4d91-860b-bb3ff61bc457`
 - a. Windows 11: `powercfg /setactive e9a42b02-d5df-448d-aa00-03f14749eb61`

(GUID identifikátory napájecích plánů jsou z interního zdroje mého zaměstnavatele)

Dalším úkonem je vypnutí animací a grafických efektů:

1. Stisknout klávesovou zkratku: Windows + S
2. Do vyhledávání zadat výkon
3. Zvolit „Upravit vzhled a výkon systému“
4. Zvolit „Optimalizovat pro výkon“
5. Zaškrtnout „Vyhladit hrany zobrazovaných písem“
6. Použít
7. OK

Posledním krokem, který nyní doporučuji je provést restart systému příkazem:
`shutdown /g /t 0 /f`

Nyní je vše připraveno k aktivaci systému a instalaci všech chybějících kumulativních aktualizací a aktualizací .NET. Rovněž je vhodné nezapomenout aktualizovat aplikace ze služby Microsoft Store.

Další optimalizaci systému Windows pro hladší běh na Raspberry Pi je možné provést například pomocí PowerShellového skriptu, který jsem zveřejnil na mém webu:
<https://elektrotechnika.vesele.info/rubriky/vyvoj-pocitacu/windows-10/aktualizovana-verze>

5.2. Aplikační kompatibilita

Na rozdíl od první mezi běžné uživatele více rozšířené desktopové verze Windows určené pro ARM, tedy Windows 8 RT, jsou na tom Windows 10 on ARM výrazně lépe pokud jde o aplikační kompatibilitu. Díky emulaci zvládají Windows 10 on ARM prakticky bez problémů pracovat s aplikacemi určenými pro x86 (32 bit Windows), což se netýká plnohodnotné verze Visual Studia.

Pokud jde o Visual Studio, je možné instalovat a provozovat VS Code ve verzi ARM64. Z plnohodnotného Visual Studia je možné využívat ARM64 nástroje pro vzdálené ladění kódu. Aplikace od Sysinternals jsou na tom dost podobně, dočkaly se i verze pro ARM, tudíž v emulaci odmítnou běh. Pokud nechce uživatel řešit, v jaké verzi spustit nástroj od Sysinternals, doporučuji, aby si sadu těchto nástrojů stáhl z MS Store.

Windows 10 on ARM nepodporují emulaci x64 aplikací, 64 bit aplikace musí být určené pro ARM64. Při pokusu o spuštění x64 aplikace se ve Windows 10 objeví dialog, který odkazuje na nalezení aplikace v MS Store, nebo nutnost použití její x86 varianty. Osobně jsem se přesvědčil, že Windows 11, právě jedenáctky tuto bariéru odbourávají, protože umí emulovat x64 aplikace, respektive jejich sdílené komponenty. Detailnější informace nabízí blog Windows.com (*Announcing ARM64EC: Building Native and Interoperable Apps for Windows 11 on ARM* [online]. Redmond: Microsoft, 2021 [cit. 2021-10-27]. Dostupné z: <https://blogs.windows.com/windowsdeveloper/2021/06/28/announcing-arm64ec-building-native-and-interoperable-apps-for-windows-11-on-arm/>).

5.3. Výkonové srovnání Windows on ARM a Windows x64

Počítač	Typ disku	Windows	CPU [%]	GPU [%]	DISK [%]	RAM [%]
TihnkServer TS150	SSD	10 x64	73,9	0,1	53,5	68,8
Surface Pro 7	SSD		45,6	4,5	232,3	54,6
Hp elitebook 8570p	SSD	11 x64	58,9	2,8	46,1	53,4
Surface Notebook 4	SSD	10 x64	51,2	18,3	97,2	113,1
RPi4	SSD	10 ARM	30,8		73,6	10,9
	HDD		26,3		31,4	10,5
	SD karta		30,5		8,3	10,6
RPi4	SSD	11 ARM	25,6		49,7	10,1
	HDD		30,2		25,6	10,7
	SD karta		30,3		7,5	10,5

Tab. 8: Výsledky testu UserBenchmark v procentuálních souhrnech – systémy obsahují kumulativní aktualizaci 2021-10

Systém	Uložiště	Studený start na plochu [s]
Windows 10	HDD	85,3±0,2
	SSD	56,2±0,2
	SD	126,9±0,2
Windows 11	HDD	171,1±0,2
	SSD	58,5±0,2
	SD	365,2±0,2
Raspberry Pi OS 32 bit	SSD	14,2±0,2
	SD	29,5±0,2

Tab. 9: Srovnání startovacích časů od zobrazení loga Raspberry Pi po plně načtenou plochu Windows on ARM³

Pět sekund za každého naměřeného času Windows jde na vrub UEFI a mají za úkol umožnit přechod do setupu. Windows 11 trvá déle přihlásit uživatele, zde je vidět posun ve zpracování Computer Configuration směrem k načítání profilu uživatele.

Test GPU pod Windows 10 skončil chybovým hlášením a ani se nespustil, na Windows 11 proběhl, ale nedokázal sebrat validní výsledky. Všechny systémy byly nastaveny na 1080p při testování, aby se dalo vše dobře srovnat.

³ Pro měření časů Bootování má uživatelský účet odstraněné heslo a systém neprovádí aktualizaci, nebo první spuštění

5.3.1. Konfigurace zařízení

Počítač	Disk	CPU	GPU	RAM
TihnkServer TS150	Intel SSDSC2BB600G4L 600GB	Intel Xeon E3-1245 v5	AMD Radeon X300	DDR3 32 GB 2133 MHz
Surface Pro 7	HFB1M8MQ331C0MR	Intel Core i3 1005G1	Intel UHD Graphics	LPDDR4X 4 GB 3733 MHz
Hp elitebook 8570p	Patriot Burst 240GB	Intel Core i7 3520M	Intel HD 4000	DDR3 8 GB 1600 MHz
Surface Notebook 4	Samsung MZ9LQ512HALU-00000	Intel Core i5 1145G7	Intel Iris Xe Graphics	LPDDR4 8 GB 4260 MHz
RPi 4	Různý, viz test ⁴	Broadcom BCM2711		LPDDR4 4GB 3200 MHz

Tab. 10: Konfigurace měřených PC

5.3.2. Subjektivní srovnání na Raspberry Pi 4

Čistě subjektivně si systém běžící na Raspberry Pi 4 neškrtně proti srovnávaným zařízením na platformě x64. Na druhou stranu dvě ze srovnávaných zařízení jsou referenční zařízení Windows 10, jedno má serverový HW včetně řadičů a poslední je sice již starý kus HW, ale ve své době představoval špičku mezi notebooky HP.

Z krátkého srovnání musím říct, že nejlépe si vedou Windows 10 na kvalitní rychlé SD kartě. Testované externí SSD připojené pomocí USB 3 (čtečka disponuje USB 3.1) je opravdu hodně hbité a vynikají výkonové problémy procesoru, který se stává velmi problémovým úzkým hrdlem. Načítání aplikací z SSD je bleskové, instalace aktualizací Windows update je obdobně svižná jako na TS120. Problémem při využití SSD je, že pohyb myši je hodně trhaný a nepřesný při přechodu mezi aplikacemi, dočítání webové stránky apod. Celkově jsou problémy s ovládáním znatelnější, pokud běží MS Edge v továrním nastavení bez blokace reklam, to pak je problém i pomocí kontextového menu vyvolat správce úloh. Uživatelský zážitek je výrazně nekonzistentní a méně příjemný nežli na většině pomalejších PC, kdy je úzkým hrdlem disk.

⁴ SSD: Patriot Burst 960 GB; R/W: 560/540 MB/s

HDD: WD Scorpio Black 7200 otáček

SD Windows 10: Kingston CANVAS Go! Plus; U3V30

SD Windows 11 a Raspberry Pi OS: Kingston microCD HC; U3

Pokud desítky běží ze svižné microSD karty nejsou bržděné ze strany disku, ale ani není tak výrazným problémem pomalý procesor. Není to o tom, že by systém nešlo využívat, systém je celkově malinko pomalejší, ale celkový uživatelský zážitek lehce vyváženější, a proto i trochu příjemnější nežli s užitím SSD. K trhání pohybu myši a jeho nepřesnosti při více oknech dochází také, nicméně celkově je systém línější.

Desítky na HDD jsou o něco plynulejší nežli na SSD při připojení stejnou čtečkou, což je paradox, protože na běžných PC je opačně. Disk tolik nepředbíhá procesor v oblasti výkonu. Problémy s trháním myši, její nepřesností a zpožděných reakcí na kliknutí tady jsou také, avšak nejsou tak markantní jako v případě použití SSD. Nutno říct, že HDD je možné použít pouze v případě, kdy je napájeno vlastním zdrojem, tedy užití běžného externího HDD formátu 2,5“ nepřipadá v úvahu, systém s těžší nastartuje a pracovat se s ním vůbec nedá.

Windows 11 naopak v kombinaci s SSD a externě napájenou čtečkou disků mile překvapily, systém je prakticky plynulý, určité zaškobrtnutí myši při práci s MS Edge se objevují, ale jsou méně rušivé nežli v případě desítek. CPU málo kdy klesne pod zcela rovnou čáru v plném výkonu, zde by opravdu pomohlo přetaktování, ale osobně jsem byl chováním jedenáctek mile překvapen. Rovněž na HDD si jedenáctky vedou dobře, jde vidět, že 7200 otáček WD Scorpio Black poskytuje dostatek výkonu uložiště na to, aby disk představoval obdobný problém, jako CPU. Systém je celkově pomalý, možná o něco málo pomalejší v instalaci aktualizací nežli při využití SSD, ale uživatelský zážitek zůstává relativně vyvážený.

5.4. Spuštění UWP využívajících GPIO

Tato možnost skutečně existuje, ale má jeden velký háček ohledně licencování systému Windows. V článku „Povolení přístupu k GPIO, I2C a SPI v uživatelském režimu“ je možné se dočíst, že je potřeba využít edici Enterprise, která stejně jako IoT edice obsahuje ovladač RhProxy sloužící k zpřístupnění GPIO, I2C a SPI aplikacím. Zde je druhý problém užití GPIO na Pi4, protože se v licencování dostáváme od bezplatné verze Windows 10 pro domácí kutily ke korporátní verzi, která představuje funkční i cenový top Windows 10. Paradoxem zůstává, že Windows 10 IoT Core obsahovaly stejný ovladač. (*Enable user mode access to GPIO, I2C, and SPI* [online]. Redmond: Microsoft, 2020 [cit. 2021-10-23]. Dostupné z: <https://docs.microsoft.com/en-us/windows/uwp/devices-sensors/enable-usermode-access#verify-controller-drivers>)

Druhým určitým háčkem, který není dokumentován je fakt, že UWP musí být buildována v konfiguraci ARM, nikoli ARM64, která odpovídá architektuře OS. Pokud se vývojář pokusí nasadit aplikaci využívající GPIO sestavenou pro ARM64, dojde k pádu aplikace a v aplikačním logu systému se dozví o chybějící komponentě: C:\Windows\System32\Windows.UI.Xaml.dll v patřičné verzi dle SDK aplikace. Knihovna Windows.UI.Xaml.dll slouží práci s grafickým rozhraním UWP, proto chyba v eventlogu může být matoucí. S nutností využít ARM místo ARM64 pro build aplikace rovněž souvisí problém s jejím debugováním, protože nástroj pro vzdálené ladění aplikací Visual studia 2019 (starší použít nelze) pro Windows on ARM je schopen debutovat pouze ARM64. Aplikaci nástroj správně nasadí a zaregistruje pro běh, správně zapíše její závislosti na systémové balíčce, vše zkolabuje až v okamžiku načítání debuggeru. Nicméně aplikace je korektně nasazena a je možné ji spustit, případné pády a další chyby odchytit logováním do souboru nebo Eventlogu.

Pokud se vývojář napíše aplikaci přímo určenou pro Windows IoT, tedy pro Windows IoT Enterprise, dostane od systému se řadu chyb, v ladícím módu Visual Studia 2019 je možné se dočíst o chybě: „DEP0700: Aplikaci se nepodařilo zaregistrovat. [0x80073CFD] Windows cannot install package Microsoft.SDKSamples.Gpio.CPP_1.0.0.0_arm__8wekyb3d8bbwe because this package is not compatible with the device. The package can only be installed on the following device families: Windows.IoT Gpio“. Chybu lze řešit, je potřeba v XLM souboru appxmanifest, který musí obsahovat následující (na rozdíl od Windows 10 IoT Core musí být uvedeny minimální a maximální verze dle SDK, nikoliv ve formátu 10.0.0.0):

<Dependencies>

```
<TargetDeviceFamily Name="Windows.Universal" MinVersion="10.0.10240.0"
MaxVersionTested="10.0.19041.0"/>
```

</Dependencies>

Dalším krokem po úpravě appxmanifestu pro úspěšný provoz aplikace je ověřit běh ovladače (služby) rhproxy:

1. Musí existovat služba rhproxy (dostupná v Enterprise edici)
2. Služba rhproxy musí být nastavena na automatické spouštění (Get-Service rhproxy | Set-Service -StartupType Automatic)

V tomto stavu již bude aplikace funkční až na pár drobností, které vyvolávají stejnou chybu jako překlad na ARM64. Z vlastní zkušenosti mohu říct, že touto chybou končí pokus o čtení z GPIO pinu na hranu, tedy jinou funkcí nežli Read() a snaha o čtení pomocí systémového přerušování změnou stavu pinu. Závažnost je stejně jako v předešlém případě v tom, že aplikační log hlásí chybu související s grafickým rozhraním, nikoliv s přístupem k GPIO. Potvrzením tohoto tvrzení je fakt, že aplikace pracující pouze na SW úrovni jako opravdu universální pro zařízení s Windows stačí jen nasadit bez jakýchkoliv úprav a kontrol.

5.5. Srovnání Windows 10 on ARM s Windows 10 IoT Core a Raspberry Pi OS při práci s GPIO

V této kapitole porovnávám Windows 10 IoT Core z prvního pololetí 2016 s Windows 10 21H1 nativní Linuxovou distribucí. Liší se i HW, systémy aktuální v roce 2021 běží na odpovídající verzi Raspberry Pi, tedy na Raspberry Pi 4 (verze se 4 GB RAM) s aktuálními ovladači i firmware k říjnu 2021. Orientační protivahou budou výsledky z Pi2 a Pi3 na Windows 10 IoT Core. Toto srovnání si dovoluji, protože Windows on ARM jsou plnohodnotným desktopem a aktualizovaná verze kódu opět vychází z aktuálního vzorového řešení Microsoftu, tudíž je v oblasti grafiky výrazně náročnější. HW i OS jsou ve srovnání s IoT Core navíc zatíženy spoustou systémových procesů, které souvisí s GUI, ale také s antimalwarovou ochranou či plnohodnotným MS Store. Výrazný rozdíl v zatížení je patrný např. na RAM paměti, kdy IoT Core potřebovalo desítky až stovky MB, Windows on ARM se pohybuje kolem 1,9 GB, pokud běží jen jedna určená aplikace a systémové procesy s GUI. Raspberry Pi 4 je provozováno bez přetaktování na originálních 1,5 GHz, stejně jako bez přetaktování probíhala měření v roce 2016 na starších modelech.

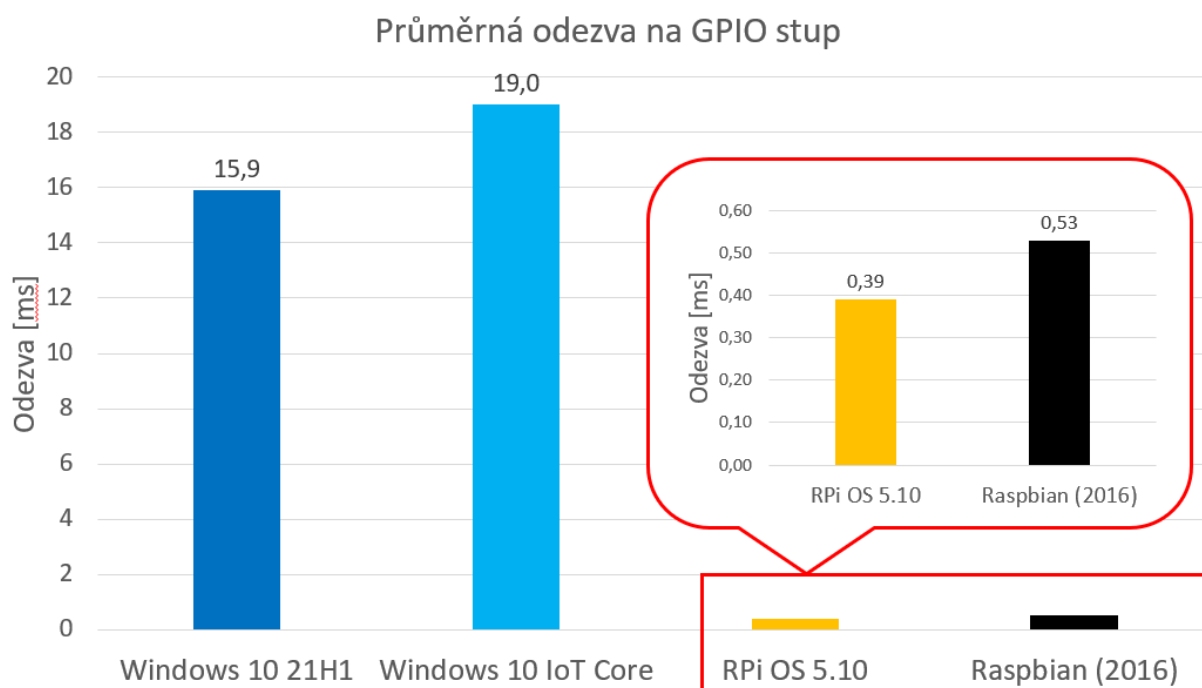
Testování probíhá obdobně jako pro Pi3 v kapitole 3.2.3 a 3.2.4. Srovnání bude mezi Windows 10 on ARM, Windows 10 IoT Core build 10.0.14342.1000 (hodnoty z Pi3 B) a Raspberry Pi OS 5.10. with Desktop ve verzi 32 bit. Všechny systémy podporované k říjnu 2021 budou měřeny na 4 GB verzi Pi4 B. Windows 11 on ARM se měření neúčastní, protože již není na nich možné spustit původní UWP s přístupem k GPIO, jedenáctky již využívají nové Windows App, které částečně vycházejí z UWP, ale nejsou vzájemně kompatibilní. Pro Windows 11 by se nejspíš musela naprogramovat aplikace v .NET 5, která by mohla být klidně konzolová.

Číslo měření	Odezva [ms]			
	Windows 10	Windows 10 IoT Core	RPI OS 5.10	Raspbian (2016)
1	13,2 ± 0,7	8,0+0,2	0,32 ± 0,02	0,54 ± 0,1
2	18,0 ± 0,9	5,2+0,1	0,33 ± 0,02	0,56 ± 0,1
3	22,0 ± 1,1	26,0+,7	0,38 ± 0,02	0,50 ± 0,1
4	20,4 ± 1,0	7,0+0,2	0,38 ± 0,02	0,47 ± 0,1
5	25,0 ± 1,3	20,0+0,5	0,39 ± 0,02	0,54 ± 0,1
6	3,4 ± 0,2	42,0+0,1	0,42 ± 0,02	0,55 ± 0,1
7	15,0 ± 0,8	18,0+0,5	0,44 ± 0,02	0,59 ± 0,2
8	26,8 ± 1,3	1,0+0,1	0,38 ± 0,02	0,50 ± 0,1
9	14,6 ± 0,7	4,6+0,1	0,42 ± 0,02	0,56 ± 0,1
10	14,2 ± 0,7	24,0+0,6	0,38 ± 0,02	0,62 ± 0,2
11	20,8 ± 1,0	14,8+0,4	0,50 ± 0,03	0,51 ± 0,1
12	2,0 ± 0,1	19,6+0,5	0,39 ± 0,02	0,51 ± 0,1
13	2,0 ± 0,1	38,0+0,5	0,40 ± 0,02	0,40 ± 0,1
14	22,2 ± 1,1	9,0+0,2	0,37 ± 0,02	0,51 ± 0,1
15	12,6 ± 0,6	55,6+1,3	0,39 ± 0,02	0,51 ± 0,1
16	12,2 ± 0,6	19,6+0,5	0,38 ± 0,02	0,43 ± 0,1
17	8,2 ± 0,4	12,2+0,3	0,38 ± 0,02	0,53 ± 0,1
18	4,6 ± 0,2	28,0+0,7	0,41 ± 0,02	0,66 ± 0,2
19	32,4 ± 1,6	11,4+0,3	0,40 ± 0,02	0,52 ± 0,1
20	29,8 ± 1,5	16,4+0,4	0,41 ± 0,02	0,50 ± 0,1
Průměr	15,9 ± 0,8	19,0+0,5	0,39 ± 0,02	0,53 ± 0,1

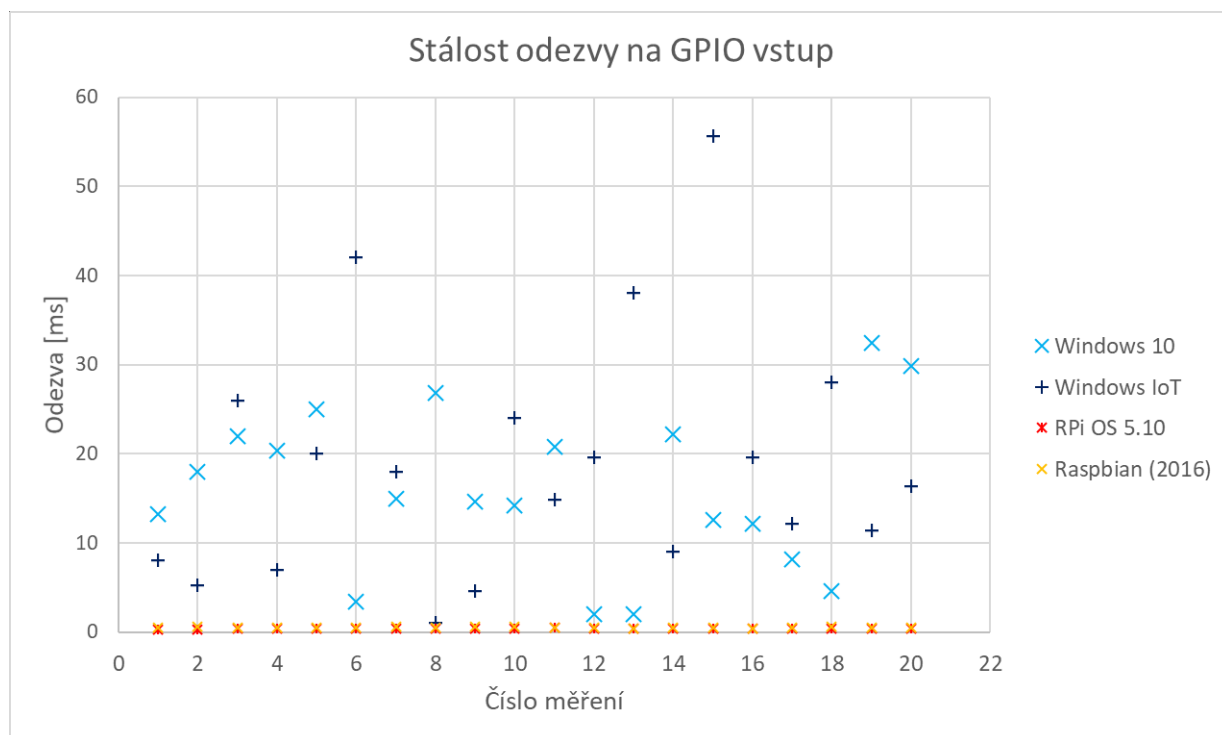
Tab. 11: Test reakční doby na GPIO

Číslo měření	Nastavená hodnota [ms]	Interval [ms]					
		Windows 10		Windows 10 IoT Core		RPi OS 5.10	
		naměřeno	rozdíl	naměřeno	rozdíl	naměřeno	rozdíl
1	0,1	15,4 ± 0,8	15,3	15,2±0,38	14,2	0,17±0,01	0,07
2	0,1	15,4 ± 0,8	15,3	16,0±0,40	15,9	0,18±0,01	0,08
3	0,1	15,6 ± 0,8	15,5	15,6±,039	15,5	0,18±0,01	0,08
4	0,1	15,6 ± 0,8	15,5	15,4±0,39	15,3	0,18±0,01	0,08
5	0,1	15,2 ± 0,8	15,1	15,4±0,39	15,3	0,18±0,01	0,08
Průměr	0,1	15,4 ± 0,8	15,3	15,5±0,39	15,4	0,18±0,01	0,08
6	1	46,0 ± 2,3	45,0	34,0±0,85	33,0	1,08±0,05	0,08
7	1	31,0 ± 1,6	30,0	32,0±0,80	31,0	1,10±0,06	0,10
8	1	47,0 ± 2,4	46,0	28,6±0,72	27,6	1,09±0,05	0,09
9	1	47,0 ± 2,4	46,0	27,6±0,69	26,6	1,08±0,05	0,08
10	1	32,0 ± 1,6	31,0	31,0±0,78	30,0	1,10±0,06	0,10
Průměr	1	40,6 ± 2,0	39,6	30,6±0,77	29,6	1,09±0,05	0,09
11	10	47 ± 2,4	37,0	35,6±0,89	25,6	10,1±0,51	0,10
12	10	46 ± 2,3	36,0	31,6±0,79	21,6	10,1±0,51	0,10
13	10	46 ± 2,3	36,0	31,8±0,80	21,8	10,1±0,51	0,10
14	10	31 ± 1,6	21,0	27,0±0,68	17,0	10,1±0,51	0,10
15	10	32 ± 1,6	22,0	27,4±0,69	17,4	10,1±0,51	0,10
Průměr	10	40,4 ± 2,0	30,4	30,7±0,77	20,7	10,1±0,51	0,10
16	500	530 ± 26,5	30,0	524,0±13,1	24,0	496,0±24,8	4,00
17	500	530 ± 26,5	30,0	512,0±12,8	12,0	500,0±25,0	0,00
18	500	540 ± 27,0	40,0	518,0±13,0	18,0	500,0±25,0	0,00
19	500	530 ± 26,5	30,0	518,0±13,0	18,0	504,0±25,2	4,00
20	500	530 ± 26,5	30,0	536,0±13,4	36,0	504,0±25,2	4,00
Průměr	500	532 ± 26,6	32,0	521,6±13,0	21,6	500,8±25,0	2,40
21	750	780 ± 39,0	30,0	788,0±19,7	38,0	750±37,5	0,0
22	750	780 ± 39,0	30,0	780,0±19,5	30,0	750±37,5	0,0
23	750	770 ± 38,5	20,0	772,0±19,3	22,0	750±37,5	0,0
24	750	780 ± 39,0	30,0	772,0±19,3	22,0	750±37,5	0,0
25	750	760 ± 38,0	10,0	780,0±19,5	30,0	750±37,5	0,0
Průměr	750	774 ± 38,7	24,0	778,4±19,5	28,4	750±37,5	0,0

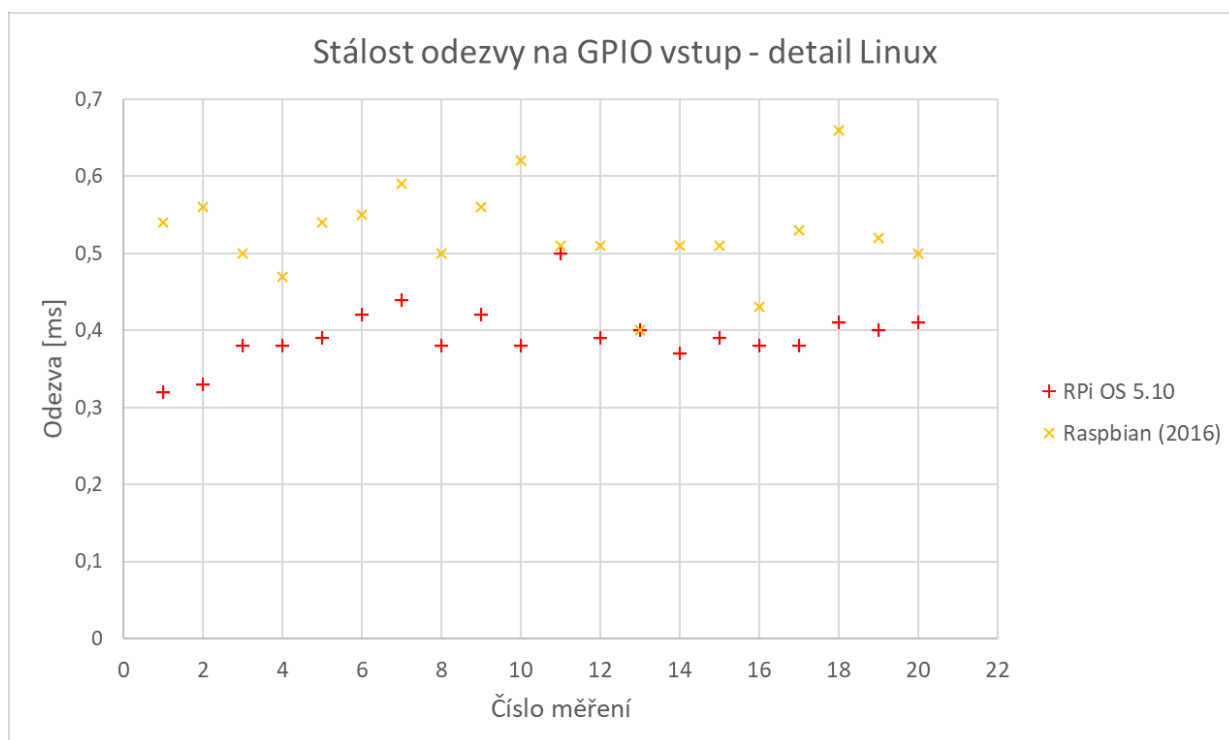
Tab. 12: Test přesnosti SW časovačů



Graf č. 4: Průměrná doba odezvy na GPIO vstup srovnání v roce 2021



Graf č. 5: Stálost odezvy na GPIO vstup



Graf č. 6: Stálost odezvy na GPIO vstup – detail nativní Linuxové distribuce

5.6. Hodnocení projektu WoR

Hodnotit projekt WoR je velmi těžké už jen proto, že nejspíš je určen pro komunitu Geeků. Za mě nedává moc smysl provozovat Desktopové Windows na Raspberry Pi, HW je na to bez dodatečný úprav prostě slabý. Pi 4B se chlubí na stránkách výrobce až 4K rozlišením na dvojici obrazovek, realita je taková, že HW má co dělat s 1080p na jedné obrazovce, proto my nedává smysl absence plnohodnotného HDMI. Stejně záškuby GUI, které vyčítám Windows, jsem registroval i na Raspberry Pi OS, který je znatelně rychlejší na kvalitním uložšti, ale trápí jej stejné problémy jako Windows 10 na SSD. Windows 11 jsou co se týká grafického rozhraní patrně nejplynulejší systém, který je možné v říjnu 2021 Raspberry Pi 4B využívat.

Pokud se budu na Pi 4B dívat jako na alternativní HW pro Windows, pak jedině pro Windows 11 on ARM, které jsou na ARM opravdu dospělým systémem, na rozdíl od adolescenta v podobě Windows 10 on ARM. V Microsoftu na systému zapracovali a díky emulaci x64 aplikací a možnosti provozu aplikací, které jsou částečně ARM64 a částečně x64 nedává před uživatele aplikační omezení, ta představuje jen HW. Na běžných PC je to přesně opačně, celkově stabilnější a dospělejší desítky mají na vrch a jediný důvod pro změnu je potřeba emulace ARM na běžném PC z důvodu vývoje. Velký problém tohoto pohledu je pro mě nutnost získat systém

pomocí neoficiálního nástroje, není možné využít Microsoft Media Creation Tool, nebo přímé stažení z Download centra Microsoftu. Ano, neoficiální program na stažení jako zdroj využívá servery služby Windows update a pro sestavení instalačního média nástroje vestavěné ve Windows, ale i tak není možné garantovat bezpečnost obrazu Windows.

Dle mého názoru je Raspberry Pi jako náhrada běžného PC příliš slabá i pro „nenáročné“ domácí uživatele, kteří využívají služeb Office 365, koukají na filmy a používají sociální sítě, právě MS Edge je tou nejproblémovější aplikací a nemyslím si, že jiný prohlížeč bude o třídu lepší na tomto HW. Ono to není dané ani tak prohlížečem jako náročností dnešní webových aplikací na klientské zařízení, proto si nemyslím, že domácí „nenáročný“ uživatel, který má běžně 2 až 20 záložek prohlížeče je cílovou skupinou. Lacinou alternativou k běžným notebookům a PC sestavám dává projekt WoR těm, kteří provozují Windows na zařízeních z éry Windows XP, nebo počátku Windows 7, u zařízení, která byla v té době již lacinějšími modely lze říct, že si uživatel nikterak nepohorší na svém zážitku a získá výhodu moderního systému Windows s pravidelnými záplatami.

Z pohledu IoT už vůbec nedává smysl provozovat desktopové Windows, takže uživatelé, kteří si pořídili RPi pro realizaci IoT řešení využijí Core verzi Linuxu, do Windows půjde spíše firemní zákazník, který si může pořídit Windows 10 IoT Enterprise, ale pro něj většinou zase není RPi vhodným HW. I s odstupem několika let trápí Windows stejné problémy s obsluhou GPIO, tím spíše velká nestabilita příliš pomalé odezvy. Kvalita SW časovačů se také ukázala jako stále silně nedostatečná na Windows a diskutabilní na Linuxu. Drobné zlepšení průměrných hodnot jak u Windows, tak u Linuxu lze přičíst na vrub silnějšímu HW, nikoliv posunu na úrovni ovladačů či samotného systému.

Pokud chci na Pi 4 provozovat aplikace napsané pro Windows 10 IoT Core, pak jsem tím správným zákazníkem projektu WoR, ale potřebuji využít Windows 10, aplikace nejsou ani po drobných úpravách kompatibilní s Windows 11. Na druhou stranu bych měl s ohledem na licenční podmínky zvážit, zda se mi nevyplatí aplikaci prostě přepsat. Většina scénářů využívajících GPIO sběrnici poběží stejně jako na nepodporovaných IoT Core a ještě přijdu o některé možnosti v oblasti zpracování signálu, které jsem na IoT Core měl k dispozici.

Je škoda, že Microsoft nechce příliš pouštět ARM verze Windows mezi lidi, osobně vím pouze o jednom zařízení, které je využívá (Microsoft Surface Pro X). Dle mého názoru by sám Microsoft měl vydat oficiální nástroje pro stažení a image ARM systému Windows pro zařízení, která na to mají a zároveň jsou lidem dostupná. Windows 11 jsou skvělou odpovědí na přechod

konkurence k ARM procesorům. Myslím si, že by to mohlo také podpořit prodeje Surface pro X, který ARM verze Windows využívá a jde o poměrně výkonné zařízení. Osobně bych Raspberry Pi nevzal vůbec do ruky, když by na něm nefungovaly Windows, jen pro mou potřebu je to příliš slabý HW a v Armových Windows chybí některé komponenty, které denně užívám (např. Hyper-V). Za mě osobně je vhodnou alternativou klasického PC nebo pracovního notebooku Surface Pro 7 (či 8), nebo Intel Nuc. Obě zařízení mají miniaturní rozměry a vysoký výkon, který člověka v práci neomezuje. Pokud jde o cenovou dostupnost, tak Intel Nuc je za mě laciným PC (s přihlédnutím k výkonu), které je snadno přenosné v kapse podobně, jako mobilní telefon.

Závěr – Výsledné hodnocení Windows 10 IoT Core

Z uživatelského pohledu se systém jeví jako velice zdařilý. Na rozdíl od většiny Linuxové konkurence nedisponuje plným GUI, které je zobrazeno přímo na připojený monitor, ale disponuje kvalitním a přehledným uživatelským rozhraním, které je dostupné prostřednictvím webserveru. Vzdálené ovládání pomocí webserveru je velmi přehledné, spolehlivé a jednoduché, mnohým lidem může připomínat portál Reporting Services z SQL serveru 2016, nebo již o něco méně podobu přístupu pomocí webového prohlížeče k operačnímu systému Windows Server 2012R2 Essentials. I instalace, především novějších buildů, je též uživatelsky velmi přívětivá, intuitivní a maximálním možným způsobem automatizovaná. Zde je na místě pochválit Microsoft za inspiraci nástroji pro instalaci mobilní verze Windows. Vzhledem ke specializaci systému si myslím, že je uživatelsky velmi přívětivý a zdařilý, jen by se dala vytknout jedna věc a to ta, že správa uživatelských účtů, jde udělat jen pomocí PowerShellu, nikoli graficky. Dle mého názoru však IoT OS nepotřebuje více jak jeden uživatelský účet.

Systém není určen ani jako odlehčená náhrada desktopových Windows, jak je to možné vidat u jeho linuxových konkurentů, ale i s ohledem na velmi rychlý vývoj se systém jeví jako obstojná volba pro projekty domácí automatizace, či jinou kutilskou činnost. Edice IoT Core se také ukázala být dobrou alternativou pro řízení informačních stánků, či tabulí, kde dokážou provozovat v individuálně upraveném prohlížeči příslušnou webovou aplikaci, nebo využít původní zdrojový balíček aplikace určené do Windows Store, což podstatně zjednodušuje údržbu i aktualizace SW vybavení takovýchto míst. V tomto směru testy prokázaly tvrzení Microsoftu o tom, k čemu je vhodná edice IoT Core.

Systém je poměrně hodně háklivý na rychlost microSD karty (má zásadní vliv na stabilitu systému), stálost napájení a postup vypínání systému (buď pomocí příkazu, pomocí webserveru, nebo pomocí výchozí aplikace systému). Při předčasném odpojení systému od napájení dochází především u Insider buildů k problémům s následujícím startem, který nezřídka končí „modrou smrtí“ a někdy se stane, že se problém nepodaří systému vyřešit a je nutné jej znovu instalovat. Systém, stejně jako celý HW, není vhodný pro běh s bateriovým napájením.

Z provedených testů je dobře patrné, že stabilita a rychlost celého řešení je hodně závislá na kvalitě aplikace, ale také na HW, na kterém běží. Mezi počítači Pi2 a Pi3 jsem pozoroval rozdíly v chování systému, které nebyly vždy předpokladatelné, nebo logické.

Velkým zklamáním bylo chování GPIO sběrnice, jejíž reakce je výrazně pomalejší, nežli pokud je na zařízení instalován Raspbian, což je v rozporu se snahou Microsoftu. Pomalé reakce GPIO sběrnice se mně osobně jeví, jako jeden ze silných omezujících faktorů, který limituje scénáře, kde lze systém využít. S GPIO souvisí také SW časovače, které byly též zklamáním, protože rozumně akceptovatelnou odchylku v kladném směru od požadované hodnoty začíná dosahovat až pro časy kolem $\frac{3}{4}$ s. Pro kutilské aplikace, kde stačí data z nějakého snímače načíst jednou za sekundu, nebo i po delších časových intervalech nejde o problém, překážkou se to stává, pokud pomocí SW časovače mají synchronizovat nějaká zařízení. Zde se dle mého názoru Microsoftu nepodařilo vyřešit programovou cestou problém s nedostatkem HW časovačů na většině zařízení, pro která je systém určen.

Co se Microsoftu povedlo opravdu splnit je, že aplikace jsou bez nutnosti dopisu jediné čárky kódu přenositelné napříč všemi zařízeními, která používají Windows 10, nebo Windows Server 2016⁵. Jediné, na co je potřeba si dát pozor je, pokud aplikace využívá specializované sběrnice (GPIO, I²C...), aby zařízení, na kterém je aplikace spouštěna tyto sběrnice opravdu fyzicky mělo a obsahovalo patřičné ovladače, což limituje přenositelnost kódů určených IoT na zbytek platformy (musí se na to myslet především při prvních pokusech ladění kódu), nicméně opačně to neplatí. Pokud si napíši například multimediální centrum, bude mi fungovat jak na IoT zařízeních (pokud bude dostačovat paměť), tak na telefonu, tabletu i počítači. Jako velké plus bych to viděl zejména pro vývojáře, kteří mohou na tomto systému testovat aplikace bez toho, aby nějak poškodili svůj vlastní OS skrytými vadami v aplikaci.

Závěrem mohu konstatovat, že Windows 10 IoT Core byl v nemalém procentu případů kvalitní bezplatnou alternativou pro scénáře, které využívají Windows Embedded a navzdory relativně rané fázi vývoje jsou kvalitním konkurentem IoT distribucím Linuxu.

⁵ Za předpokladu, že systém je vybaven Desktop Experience, uživatel není built in admin a nemá zakázané UWP bezpečnostní politikou.

Pohled z roku 2021

Systém navzdory svým kvalitám v určité oblasti nedosáhl takového rozšíření a objemu čerpání služeb Azure, aby splnil obchodní záměr vydavatele. Provozování systému je dnes možné jen na starším HW, který je ovšem pro většinu scénářů více než dostačující. Nedostatky v oblasti SW časovačů či odezvy na události GPOI se nevyřešily (soudě dle využití stejného ovladače a služeb v rámci Windows 10 on ARM). Problémem s provozováním systému po ukončení jeho podpory vím v tom, že chybí pravidelné bezpečnostní aktualizace a rovněž vývojové nástroje, které jsou pro daný systém primárně určeny se blíží konci svého životního cyklu. Navzdory určitým výhodám bych dnes nemohl systém doporučit pro provoz kioskových řešení právě z důvodu bezpečnosti.

Právě rok 2019 je velkým milníkem v životě Windows 10, do té doby skutečně multiplatformní systém se opět stává desktopovým a serverovým operačním systémem, stejně jako jeho předchůdci. Do oblasti „domácího“ IoT i světa smartphonů Windows vstoupily jako poslední, bohužel se jim nepodařilo získat dostatečný tržní podíl na to, aby byly pro Microsoft rentabilní. Dle mého názoru je to velká škoda, protože jak mobilní, tak IoT Core verze byly kvalitní a zanechaly po sobě řadu odkazů v podobě dílčích celků či inspirace, které nalézáme v jiných produktech Microsoftu, nebo i konkurenčních řešeních. Druhým úhlem pohledu si dovolím tvrdit, že mezi lety 2016 a 2019 byly Windows 10 z pohledu multiplatformnosti lepší, minimálně srovnatelné, s rodinou operačních systémů společnosti Apple. Na rozdíl od Apple neměly desítky nikdy dedikovanou verzi pro SmartTV, ale měly edici pro IoT, kterou rodina operačních systému Apple postrádala. Obdobné aplikační jednotnosti se rodina operačních systému od Apple dočkala až s příchodem procesoru M1.

Porovnat desítky s Linuxem z pohledu multiplatformnosti je dalece těžší úkol, osobně neznám konkrétní distribuci, která by pokrývala od mobilních telefonů přes IoT a desktopy až po servery. Kolem v roce 2015 se desítkám snažilo přiblížit Ubuntu, avšak jeho mobilní verze zůstala jen pro vyvolené v jediném kusu HW s kratší životností, nežli měly Windows 10 Mobile. Pokud se ovšem podívám na Linux ne jako na konkrétní distribuce, ale jako na samotné jádro, pak rozhodne nad desítkami vede, už jen z toho důvodu, že je využíván v routerech a Android, nejrozšířenější mobilní operační systém, stojí právě nad jádrem Linuxu. Pohled na jádra OS ještě umocňuje fakt, že Microsoft si vyvíjí vlastní distribuci Linuxu, kterou využívá s Load balancery a síťových prvcích pro Azure, dále pak pro vybrané scénáře služby Azure IoT Hub, nebo jako základ subsystému for Linux ve Windows 10 a 11.

Celkově si myslím, že běh systému Windows na HW, který je určen pro „domácí“ IoT bez ohledu na projekt WoR není dobrým nápadem. Tato zařízení nemají sloužit jako náhrada počítače, byť k tomu mohou parametry Pi 4B svádět. Konkrétnější závěry jsou v kapitole 5.6. Ostatně sám Microsoft dnes preferuje cestu multiplatformního .NET a využití Dockeru pro IoT i nemalou část produktů kategorie Edge Computing.

Odpověď na to, zda byly desítky opravdu multiplatformním systémem zní ANO, bohužel tomu tak bylo jen do konce roku 2019. S Windows 10 verze 1809 a novější udělal Microsoft krok zpět, pokud jde o Windows, obchodně to ale byl díky .NET 5 spíš krok vpřed na poli služeb.

Seznam použité literatury

Publikace

1. Petr Roudenský, Anna Havlíčková. *Řízení kvality softwaru*. Brno : Computer Press, 2013. IBSN 978-80-251-3816-8.
2. VÍT, Tomáš. Raspberry Pi 2: Ještě rychlejší minipočítač. *Computer*. Brno: Mladá fronta, 2015, **2015**(4), 3.
3. *The MagPi: The official Raspberry Pi magazine*. 2016, **2016**(8).

Internetové zdroje

4. PowerShell to connect to device running Windows 10 IoT Core Core. *Hackster.io*. [Online] Microsoft, 2015. [Citace: 13. 11 2015.]
<https://www.hackster.io/windowsiot/powershell-to-connect-to-a-machine-runningwindows-10-16a479>.
5. Deploying an App with Visual Studio. *Windows Dev Center*. [Online] Microsoft. [Citace: 24. 7 2016.]
<https://developer.microsoft.com/enus/windows/iot/win10/AppDeployment.htm#python>.
6. Windows 10 for the Internet of Your Things. *Microsoft*. [Online] Microsoft, 2015. [Citace: 13. 11 2015.]
<https://www.microsoft.com/enus/WindowsForBusiness/windows-iot>
7. Porovnání nabídek nástroje Visual Studio 2015. *Visual Studio*. [Online] Microsoft. [Citace: 29. 5 2016.] <https://www.visualstudio.com/cs-cz/products/compare-visualstudio-2015-products-vs.aspx>
8. Downloads and Tools. *Windows Dev Center*. [Online] Microsoft. [Citace: 5. 11 2015.] <https://developer.microsoft.com/en-us/windows/iot/Downloads.htm>.
9. Soubory ke stažení pro Visual Studio. *Visual Studio*. [Online] Microsoft. [Citace: 20. 11 2015.] <https://www.visualstudio.com/cs-cz/downloads/download-visualstudio-vs.aspx>.
10. Guids. *Windows on Raspberry* [online]. [cit. 2021-10-26].
<https://www.worproject.ml>

Sezam použitých obrázků

- Obr.1: Obsah instalátoru Windows 10 IoT Core
- Obr. 2: Obsah souboru File_WindowsIoTRpi2.ffu
- Obr. 3: Vybrání souboru obrazu systému, který bude instalován
- Obr. 4: Program dism.exe
- Obr. 5: Načítání aplikace na Windows 10 IoT Core
- Obr. 6: Pomocná aplikace konfigurace OS
- Obr. 7: Výchozí aplikace, pokud je systém připojen k Wi-Fi i Ethernetu
- Obr. 8: WindowsIoTCoreWatcher
- Obr. 9: Přihlášení pomocí PowerShellu k Windows 10 IoT Core
- Obr. 11: Vypnutí počítače Raspberry pomocí PowerShellu
- Obr. 12: Přihlášení k Windows 10 IoT Core pomocí Microsoft Edge
- Obr. 13: Úvodní stránka webového rozhraní
- Obr. 14: Stránka Apps
- Obr. 15: Stránka procesů
- Obr. 16: Stránka Performance
- Obr. 17: Stránka pro konfiguraci síťového připojení
- Obr. 18: Výstup příkazu Get-Process
- Obr. 19: Výpis IP konfigurace systému
- Obr. 20: Zobrazení architektury procesoru pomocí PowerShell
- Obr. 21: Přehled parametrů příkazu „reg.exe“
- Obr. 22: Aktivace vývojářského režimu v systému Windows 10
- Obr. 23: Založení projektu Windows Universal Apps ve Visual Studiu
- Obr. 24: Postup přidání referencí do projektu Visual Studia
- Obr. 25: Přidání referencí pro Windows IoT
- Obr. 26: Grafický editor souboru MainPage.xaml
- Obr. 27: Vložení textového bloku v návrhu MainPage
- Obr. 28: Spuštění ladění ve Visual Studiu
- Obr. 29: Aplikace Ahoj světe při ladění
- Obr. 30: Visual Studio při ladění aplikace Ahoj světe
- Obr. 31: Nastavení režimu vzdáleného ladění
- Obr. 32: Nastavení vzdáleného ladění

- Obr. 33: Vzdálené ladění na zobrazovači Raspberry Pi
- Obr. 34: Konfigurace aplikace pro umístění do Store
- Obr. 35: Otevření průvodce pro tvorbu balíčku Windows universal Apps
- Obr. 36: Vytvoření balíčku aplikace krok 1
- Obr. 37: Vytvoření balíčku aplikace krok 2
- Obr. 38: Vytvoření certifikátu aplikace
- Obr. 39: Nastavení automatizovaných testů aplikace při generování certifikátu
- Obr. 40: Možnosti připojení se k Windows 10 IoT v aplikaci IoT Dashboard
- Obr. 41: Instalace systému pomocí IoT Dashboardu
- Obr. 42: CPU trace z běhu aplikace teploměr na Pi3
- Obr. 43: Memory a Disk trace z běhu aplikace Teploměr na Pi3
- Obr. 44: CPU trace z běhu aplikace Teploměr na Pi2
- Obr. 45: Memory a Disk trace z běhu aplikace Teploměr na Pi2

Seznam Tabulek

Tabulka 1: HW požadavky Windows 10 pro stolní počítač

Tabulka 2: Požadavky Visual Studio 2015 Professional

Tab. 3: Vysvětlení zkratk ve výpisu příkazu Get-Process

Tab. 4: Časy načítání webových služeb a stránek

Tab. 5: Naměřené časy odezvy

Tab. 6: Naměřené hodnoty přesnosti časovače Pi2

Tab. 7: Naměřené hodnoty přesnosti časovače Pi3

Tab. 8: Výsledky testu UserBenchmark v procentuálních souhrnech – systémy obsahují kumulativní aktualizaci 2021-10

Tab. 9: Srovnání startovacích časů od zobrazení loga Raspberry Pi po plně načtenou plochu Windows on ARM

Tab. 10: Konfigurace měřených PC

Tab. 11: Test reakční doby na GPIO

Tab. 12: Test přesnosti SW časovačů

Seznam grafů

Graf č. 1: Srovnání průměrné rychlosti odezvy Windows a Raspbianu na Pi2 a Pi3

Graf č. 2: Stálost odezvy Windows 10 IoT Core

Graf č. 3: Stabilita odezvy Raspbian

Graf č. 4: Průměrná doba odezvy na GPIO vstup srovnání v roce 2021

Graf č. 5: Stálost odezvy na GPIO vstup

Graf č. 6: Stálost odezvy na GPIO vstup – detail nativní Linuxové distribuce

Seznam použitých zkratek

Zkratka	Význam zkratky
UWP	Univerzální aplikace Windows 10
UAI	Ústav aplikované informatiky Přírodovědecké fakulty Jihočeské univerzity v Českých Budějovicích
UFY	Ústav fyziky a biofyziky Přírodovědecké fakulty Jihočeské univerzity v Českých Budějovicích
Mgr.	Magistr, akademický titul
Ph.D.	Vysokoškolský titul
Sb.	Sbírka, ve významu sbírky zákonů
odst.	Odstavec
JCU	Jihočeská univerzita v Českých Budějovicích
č.	Číslo
IoT	Internet of Things (internet věcí)
SW	Software
HW	Hardware
OS	Operační systém
ATM	Bankomat, prodejní terminál
Tzv.	Takzvaný
Např.	Například
RTM	Release To Manufacturing (produkční verze)
U1	Rychlostní standard paměťových karet
Wi-Fi nebo WiFi	Wireless Fidelity (standard bezdrátových sítí)
USB	Universal Serial Bus
GPIO	General-purpose input/output
x86	32 bitová architektura operačního systému či procesoru
x64	64 bitová architektura operačního systému či procesoru
ARM	Advanced RISC Machine
TP3	Technical Preview 3 (samotné TP je pak Technical Preview)
HDD	Pevný disk
RAM	Paměť s libovolným přístupem (vyrovnávací paměť počítače)
PC	Osobní počítač, v širším významu jen počítač
IDE	Integrované vývojové prostředí

Zkratka	Význam zkratky
MSDN	Microsoft Developers Network
MSDN AA	MicroSoft Developers Network for Academics alliance
DVD	Digital Video Disk
HDMI	High-Definition Multi-media Interface
SDHC	Secure digital High capacity
Apps	Aplikace moderního uživatelského rozhraní Windows 8, 8.1, Server 2012 a Server 2012R2, pokud předchází slovo Universal, jedná se o aplikace pro rodinu systémů Windows 10 a Windows Server 2016
IP	Internet Protocol
MAC	Media Access Control
SSH	Secure Shell
NT	Typ jádra OS Windows (New Technology)
NOOBS	New Out Of the Box Software
CPU	Central Procesor Unit
LED	Dioda emitující světlo na principu spontánní emise elektronů

Seznam příloh

- A. Příkazy vzdálené relace Windows 10 IoT Core začínající klíčovým slovem get
- B. Popis pinů Raspberry Pi 2
- C. Zdrojový kód testování reakční doby
- D. Zdrojový kód testu přesnosti SW časovačů
- E. Zdrojový kód informačního stánku
- F. Zdrojový kód webového prohlížeče
- G. Ukázkový zdrojový kód v Pythonu
- H. Zdrojový kód měření reakční doby GPIO sběrnice pro Raspbian
- I. Obrazová dokumentace měření na GPIO sběrnici
- J. Obrázky novinek ve webovém rozhraní pro správu OS
- K. Obrázky testu práce s internetem
- L. Postup stažení systému pro Pi3
- M. Grafy vytížení systému při simulaci informačního stánku
- N. Práce s FTP

Příloha A: Příkazy vzdálené relace Windows 10 IoT začínající klíčovým slovem get

<i>Name</i>	<i>Category</i>	<i>Module</i>	<i>Synopsis</i>
Get-Verb	Function		
Get-FileHash	Function	Microsoft.PowerShell.U...	...
Get-Command	Cmdlet	Microsoft.PowerShell.Core	...
Get-Help	Cmdlet	Microsoft.PowerShell.Core	...
Get-History	Cmdlet	Microsoft.PowerShell.Core	...
Get-Job	Cmdlet	Microsoft.PowerShell.Core	...
Get-Module	Cmdlet	Microsoft.PowerShell.Core	...
Get-PSHostProcessInfo	Cmdlet	Microsoft.PowerShell.Core	...
Get-PSSession	Cmdlet	Microsoft.PowerShell.Core	...
Get-PSSessionCapability	Cmdlet	Microsoft.PowerShell.Core	...
Get-PSSessionConfiguration	Cmdlet	Microsoft.PowerShell.Core	...
Get-Alias	Cmdlet	Microsoft.PowerShell.U...	...
Get-Culture	Cmdlet	Microsoft.PowerShell.U...	...
Get-Date	Cmdlet	Microsoft.PowerShell.U...	...
Get-Event	Cmdlet	Microsoft.PowerShell.U...	...
Get-EventSubscriber	Cmdlet	Microsoft.PowerShell.U...	...
Get-FormatData	Cmdlet	Microsoft.PowerShell.U...	...
Get-Host	Cmdlet	Microsoft.PowerShell.U...	...
Get-Member	Cmdlet	Microsoft.PowerShell.U...	...
Get-PSBreakpoint	Cmdlet	Microsoft.PowerShell.U...	...
Get-PSCallStack	Cmdlet	Microsoft.PowerShell.U...	...
Get-Random	Cmdlet	Microsoft.PowerShell.U...	...
Get-Runspace	Cmdlet	Microsoft.PowerShell.U...	...
Get-RunspaceDebug	Cmdlet	Microsoft.PowerShell.U...	...
Get-TypeData	Cmdlet	Microsoft.PowerShell.U...	...
Get-UICulture	Cmdlet	Microsoft.PowerShell.U...	...
Get-Unique	Cmdlet	Microsoft.PowerShell.U...	...
Get-Variable	Cmdlet	Microsoft.PowerShell.U...	...
Get-ControlPanellItem	Cmdlet	Microsoft.PowerShell.M...	Get-ControlPanellItem...

Get-WmiObject	Cmdlet	Microsoft.PowerShell.M...	Get-WmiObject...
Get-Location	Cmdlet	Microsoft.PowerShell.M...	Get-Location...
Get-ComputerInfo	Cmdlet	Microsoft.PowerShell.M...	Get-ComputerInfo...
Get-PSProvider	Cmdlet	Microsoft.PowerShell.M...	Get-PSProvider...
Get-PSDrive	Cmdlet	Microsoft.PowerShell.M...	Get-PSDrive...
Get-ChildItem	Cmdlet	Microsoft.PowerShell.M...	Get-ChildItem...
Get-Content	Cmdlet	Microsoft.PowerShell.M...	Get-Content...
Get-Process	Cmdlet	Microsoft.PowerShell.M...	Get-Process...
Get-ItemPropertyValue	Cmdlet	Microsoft.PowerShell.M...	Get-ItemPropertyValue...
Get-Service	Cmdlet	Microsoft.PowerShell.M...	Get-Service...
Get-HotFix	Cmdlet	Microsoft.PowerShell.M...	Get-HotFix...

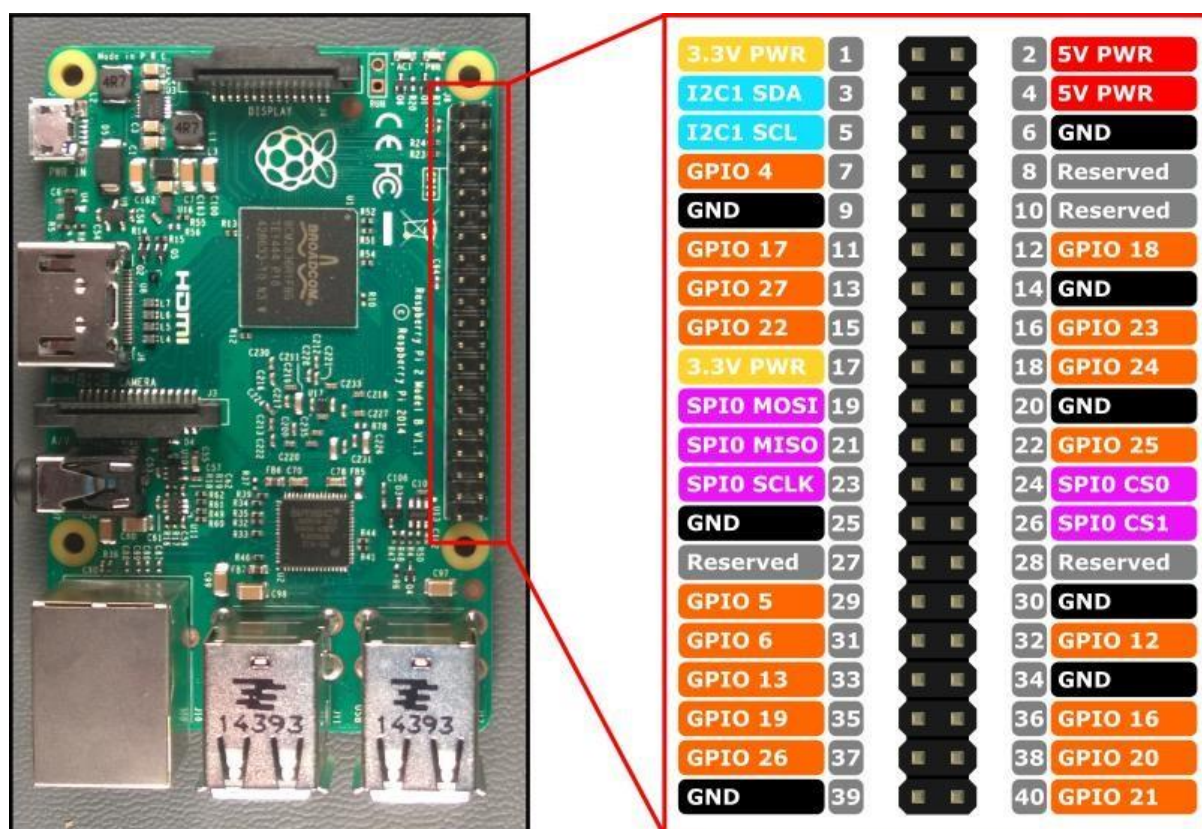
Name	Category Module		Synopsis
Get-TimeZone	Cmdlet	Microsoft.PowerShell.M...	Get-TimeZone...
Get-Clipboard	Cmdlet	Microsoft.PowerShell.M...	Get-Clipboard...
Get-EventLog	Cmdlet	Microsoft.PowerShell.M...	Get-EventLog...
Get-Transaction	Cmdlet	Microsoft.PowerShell.M...	Get-Transaction...
Get-ComputerRestorePoint	Cmdlet	Microsoft.PowerShell.M...	Get-ComputerRestorePoint...
Get-Item	Cmdlet	Microsoft.PowerShell.M...	Get-Item...
Get-ItemProperty	Cmdlet	Microsoft.PowerShell.M...	Get-ItemProperty...
Get-AppxVolume	Cmdlet	Appx	Get-AppxVolume...
Get-AppxPackageManifest	Cmdlet	Appx	Get-AppxPackageManifest...
Get-AppxLog	Function	Appx	...
Get-AppxDefaultVolume	Cmdlet	Appx	Get-AppxDefaultVolume...
Get-AppxLastError	Function	Appx	...
Get-AppxPackage	Cmdlet	Appx	Get-AppxPackage...
Get-CimInstance	Cmdlet	CimCmdlets	Get-CimInstance...
Get-CimClass	Cmdlet	CimCmdlets	Get-CimClass...
Get-CimSession	Cmdlet	CimCmdlets	Get-CimSession...
Get-CimAssociatedInstance	Cmdlet	CimCmdlets	Get-CimAssociatedInstance...
Get-DnsClientCache	Function	DnsClient	...
Get-DnsClientServerAddress	Function	DnsClient	...
Get-DnsClient	Function	DnsClient	...
Get-DnsClientGlobalSetting	Function	DnsClient	...
Get-WinEvent	Cmdlet	Microsoft.PowerShell.D...	Get-WinEvent...
Get-LocalUser	Cmdlet	Microsoft.PowerShell.L...	Get-LocalUser...
Get-LocalGroup	Cmdlet	Microsoft.PowerShell.L...	Get-LocalGroup...
Get-LocalGroupMember	Cmdlet	Microsoft.PowerShell.L...	Get-LocalGroupMember...
Get-AuthenticodeSignature	Cmdlet	Microsoft.PowerShell.S...	Get-AuthenticodeSignature...
Get-ExecutionPolicy	Cmdlet	Microsoft.PowerShell.S...	Get-ExecutionPolicy...
Get-Credential	Cmdlet	Microsoft.PowerShell.S...	Get-Credential...
Get-Acl	Cmdlet	Microsoft.PowerShell.S...	Get-Acl...
Get-WSManInstance	Cmdlet	Microsoft.WSMan.Manage...	Get-WSManInstance...
Get-WSManCredSSP	Cmdlet	Microsoft.WSMan.Manage...	Get-WSManCredSSP...
Get-NetAdapterPowerManagement	Function	NetAdapter	...

Name	Category Module	Synopsis
Get-NetAdapterIPsecOffload	Function NetAdapter	...
Get-NetAdapterQos	Function NetAdapter	...
Get-NetAdapterRdma	Function NetAdapter	...
Get-NetAdapterPacketDirect	Function NetAdapter	...
Get-NetAdapterChecksumOffload	Function NetAdapter	...
Get-NetAdapterVMQueue	Function NetAdapter	...
Get-NetAdapterRss	Function NetAdapter	...
Get-NetAdapterEncapsulatedPack...	Function NetAdapter	...
Get-NetAdapterSriov	Function NetAdapter	...
Get-NetAdapterVmq	Function NetAdapter	...
Get-NetAdapterHardwareInfo	Function NetAdapter	...
Get-NetAdapterVPort	Function NetAdapter	...
Get-NetAdapterSriovVf	Function NetAdapter	...
Get-NetAdapterLso	Function NetAdapter	...
Get-NetAdapterAdvancedProperty	Function NetAdapter	...
Get-NetAdapterStatistics	Function NetAdapter	...
Get-NetAdapterRsc	Function NetAdapter	...
Get-NetAdapter	Function NetAdapter	...
Get-NetAdapterBinding	Function NetAdapter	...
Get-NetEventProvider	Function NetEventPacketCapture	...
Get-NetEventVmSwitchProvider	Function NetEventPacketCapture	...
Get-NetEventPacketCaptureProvider	Function NetEventPacketCapture	...
Get-NetEventNetworkAdapter	Function NetEventPacketCapture	...
Get-NetEventVmNetworkAdapter	Function NetEventPacketCapture	...
Get-NetEventVmSwitch	Function NetEventPacketCapture	...
Get-NetEventVFPPProvider	Function NetEventPacketCapture	...
Get-NetEventSession	Function NetEventPacketCapture	...
Get-NetEventWFPCaptureProvider	Function NetEventPacketCapture	...
Get-NetIPsecPhase2AuthSet	Function NetSecurity	...
Get-NetFirewallPortFilter	Function NetSecurity	...
Get-NetFirewallProfile	Function NetSecurity	...

Name	Category Module		Synopsis
Get-NetIPsecMainModeSA	Function	NetSecurity	...
Get-NetFirewallInterfaceFilter	Function	NetSecurity	...
Get-NetIPsecRule	Function	NetSecurity	...
Get-NetIPsecMainModeRule	Function	NetSecurity	...
Get-NetFirewallServiceFilter	Function	NetSecurity	...
Get-DAPolicyChange	Cmdlet	NetSecurity	...
Get-NetFirewallSecurityFilter	Function	NetSecurity	...
Get-NetFirewallInterfaceTypeFi...	Function	NetSecurity	...
Get-NetFirewallSetting	Function	NetSecurity	...
Get-NetFirewallRule	Function	NetSecurity	...
Get-NetFirewallApplicationFilter	Function	NetSecurity	...
Get-NetIPsecQuickModeCryptoSet	Function	NetSecurity	...
Get-NetFirewallAddressFilter	Function	NetSecurity	...
Get-NetIPsecQuickModeSA	Function	NetSecurity	...
Get-NetIPsecDospSetting	Function	NetSecurity	...
Get-NetIPsecMainModeCryptoSet	Function	NetSecurity	...
Get-NetIPsecPhase1AuthSet	Function	NetSecurity	...
Get-NetCompartment	Function	NetTCPIP	...
Get-NetOffloadGlobalSetting	Function	NetTCPIP	...
Get-NetUDPSetting	Function	NetTCPIP	...
Get-NetUDPEndpoint	Function	NetTCPIP	...
Get-NetTCPSetting	Function	NetTCPIP	...
Get-NetTransportFilter	Function	NetTCPIP	...
Get-NetIPAddress	Function	NetTCPIP	...
Get-NetIPv6Protocol	Function	NetTCPIP	...
Get-NetIPInterface	Function	NetTCPIP	...
Get-NetRoute	Function	NetTCPIP	...

<i>Name</i>	<i>Category</i>	<i>Module</i>	<i>Synopsis</i>
Get-NetIPConfiguration	Function	NetTCPIP	...
Get-NetIPv4Protocol	Function	NetTCPIP	...
Get-NetTCPConnection	Function	NetTCPIP	...
Get-NetNeighbor	Function	NetTCPIP	...
Get-NetPrefixPolicy	Function	NetTCPIP	...
Get-PnpDeviceProperty	Function	PnpDevice	...
Get-PnpDevice	Function	PnpDevice	...
Get-TpmSupportedFeature	Cmdlet	TrustedPlatformModule	...
Get-TpmEndorsementKeyInfo	Cmdlet	TrustedPlatformModule	..
Get-Tpm	Cmdlet	TrustedPlatformModule	.

Příloha B: Popis pinů Raspberry Pi 2



Popis pinů Raspberry Pi 2 (Zdroj: Windows Dev Center: Blinky Sample [online]. 2015 [cit. 2015-11-21].
Dostupné z: <http://ms-iot.github.io/content/en-US/win10/samples/Blinky.htm>)

Příloha C: Zdrojový kód testování reakční doby

Soubor MainPage.xaml

```
<Page
    x:Class="GPIOIntroPi.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:GPIOIntroPi"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

        <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">

            <TextBlock x:Name="GpioStatus" Text="Waiting to initialize GPIO..."
                Margin="10,50,10,10" TextAlignment="Center" FontSize="26.667" />

        </StackPanel>

    </Grid>
</Page>
```

Zdroj: GitHubab: MainPage.xaml [online]. 2015, 6.7.2015 [cit. 2015-11-21]. Dostupné z: <https://gist.githubusercontent.com/ChrisBriggsy/82c7a1a3cd24e553ca29/raw/59ee5c27276e17e96b8e623c9afe013003ffc534/MainPage.xaml>

Soubor MainPage.xaml.cs

```
using
Windows.UI.Xaml.Controls;
using Windows.Devices.Gpio;
using Windows.UI.Xaml; using
System;

namespace GPIOIntroPi
{
    public sealed partial class MainPage : Page
    {
        private const int LED_PIN = 6;
        private const int PB_PIN = 5;
        private GpioPin pin; private GpioPin
        pushButton; private DispatcherTimer
        timer; private GpioPinValue
        pushButtonValue; public MainPage()
        {
            InitializeComponent();
            timer = new DispatcherTimer();
            timer.Interval = TimeSpan.FromMilliseconds(1);
            timer.Tick += Timer_Tick; timer.Start();
            Unloaded += MainPage_Unloaded;
            InitGPIO();
        }
        private void InitGPIO()
        {
```

```

        var gpio = GpioController.Default();
if (gpio == null)
{
    GpioStatus.Text = "Toto zařízení nemá GPIO.";
return;
}
pushButton = gpio.OpenPin(PB_PIN);
pin = gpio.OpenPin(LED_PIN);
pushButton.SetDriveMode(GpioPinDriveMode.Input);
pin.Write(GpioPinValue.High); pin.SetDriveMode(GpioPinDriveMode.Output);
GpioStatus.Text = "GPIO je připravena.";
}
private void MainPage_Unloaded(object sender, object args)
{
    pin.Dispose();
pushButton.Dispose();
}
private void FlipLED()
{
    pushButtonValue = pushButton.Read();
if (pushButtonValue == GpioPinValue.High)
{
    pin.Write(GpioPinValue.High);
}
else if (pushButtonValue == GpioPinValue.Low)
{
    pin.Write(GpioPinValue.Low);
}
}
private void Timer_Tick(object sender, object e)
{
    FlipLED();
}
} }

```

Upraveno: *GitHubab: MainPage.xaml [online]. 2015, 6.7.2015 [cit. 2015-11-21]. Dostupné z: <https://gist.github.com/ChrisBriggsy/d7ca4826e0919135cd16/raw/5a71e4363aebca430343d9edea3f6c13d464306/MainPage.xaml.cs>*

Soubor App.xaml.cs

```

using GPIOIntroPi; using
System;
using
System.Collections.Generic;
using System.IO; using
System.Linq;
using
System.Runtime.InteropServices.WindowsRuntime;
using Windows.ApplicationModel; using
Windows.ApplicationModel.Activation; using
Windows.Foundation; using
Windows.Foundation.Collections; using
Windows.UI.Xaml; using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives; using
Windows.UI.Xaml.Data; using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media; using
Windows.UI.Xaml.Navigation;

```

```

namespace spinani
{
    /// <summary>
    /// Poskytuje chování specifické pro aplikaci, které doplňuje výchozí třídu
    Application.
    /// </summary>
    sealed partial class App : Application
    {
        /// <summary>
        /// Inicializuje objekt aplikace typu singleton. Jedná se o první řádek
        spuštěného vytvořeného kódu,
        /// který je proto logickým ekvivalentem metod main() nebo WinMain().
        /// </summary>
        public App()
        {
            Microsoft.ApplicationInsights.WindowsAppInitializer.InitializeAsync(
                Microsoft.ApplicationInsights.WindowsCollectors.Metadata |
                Microsoft.ApplicationInsights.WindowsCollectors.Session);
            this.InitializeComponent();          this.Suspending += OnSuspending;
        }

        /// <summary>
        /// Vymolá se při normálním spuštění aplikace koncovým uživatelem. Ostatní
        vstupní body
        /// se použijí například při spuštění aplikace za účelem
        otevření konkrétního souboru.          /// </summary>
        /// <param name="e">Podrobnosti o žádosti o spuštění a procesu</param>
        protected override void OnLaunched(LaunchActivatedEventArgs e)
        {
            #if DEBUG
                if (System.Diagnostics.Debugger.IsAttached)
                {
                    this.DebugSettings.EnableFrameRateCounter = true;
                }
            #endif

            Frame rootFrame = Window.Current.Content as Frame;

            // Neopakovat inicializaci aplikace, pokud už má okno obsah,
            // jenom ověřit, jestli je toto okno aktivní
            if (rootFrame == null)
            {
                // Vytvořit objekt Frame, který bude fungovat jako kontext navigace,
                a spustit procházení první stránky          rootFrame = new Frame();
                rootFrame.NavigationFailed +=
                OnNavigationFailed;

                if (e.PreviousExecutionState ==
                ApplicationExecutionState.Terminated)
                {
                    //TODO: Načíst stav z dříve pozastavené aplikace
                }

                // Umístit rámeček do aktuálního objektu Window
                Window.Current.Content = rootFrame;
            }
        }
    }
}

```

```

    }
    if (rootFrame.Content ==
null)
    {
        // Pokud není navigační zásobník obnovený, přejít na první stránku
        // a nakonfigurovat novou stránku předáním požadovaných informací ve formě
        // parametru navigace
        rootFrame.Navigate(typeof(MainPage), e.Arguments);
    }
    // Zkontrolovat, jestli je aktuální okno aktivní
    Window.Current.Activate();
}

/// <summary>
/// Vyvolá se, když selže přechod na určitou stránku.
/// </summary>
/// <param name="sender">Objekt Frame, u kterého selhala navigace</param>
/// <param name="e">Podrobnosti o chybě navigace</param>
void OnNavigationFailed(object sender, NavigationFailedEventArgs e)
{
    throw new Exception("Failed to load Page " + e.SourcePageType.FullName);
}

/// <summary>
/// Vyvolá se při pozastavení běhu aplikace. Stav aplikace se uloží
/// bez informace, jestli se aplikace ukončí nebo obnoví s obsahem
/// stále neporušené paměti.
/// </summary>
/// <param name="sender">Zdroj žádosti o pozastavení</param>
/// <param name="e">Podrobnosti žádosti o pozastavení</param>
private void OnSuspending(object sender, SuspendingEventArgs e)
{
    var deferral = e.SuspendingOperation.GetDeferral();
    //TODO: Uložit stav aplikace a zastavit jakoukoliv aktivitu na pozadí
    deferral.Complete();
}
} }

```

Zdroj: Automaticky generováno programem Visual Studio

PřílohaD: Zdrojový kód testu přesnosti SW časovačů

Soubor MainPage.xaml

```
<!--
    Copyright(c) Microsoft Open Technologies, Inc. All rights reserved.
    The MIT License(MIT)
    Permission is hereby granted, free of charge, to any person obtaining a copy
    of this software and associated documentation files(the "Software"), to deal    in
    the Software without restriction, including without limitation the rights    to
    use, copy, modify, merge, publish, distribute, sublicense, and / or sell    copies
    of the Software, and to permit persons to whom the Software is    furnished to do
    so, subject to the following conditions :
    The above copyright notice and this permission notice shall be included in
    all copies or substantial portions of the Software.
    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.IN NO EVENT SHALL THE
    AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN    THE
    SOFTWARE.
-->

<Page
    x:Class="Blinky.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"    xmlns:local="using:Blinky"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
            <Ellipse x:Name="LED" Fill="LightGray" Stroke="White" Width="100"
Height="100" Margin="10"/>
            <TextBlock x:Name="DelayText" Text="500ms" Margin="10"
TextAlignment="Center" FontSize="26.667" />
            <TextBlock x:Name="GpioStatus" Text="Počkejte na inicializaci GPIO..."
Margin="10,50,10,10" TextAlignment="Center" FontSize="26.667" />
        </StackPanel>
    </Grid> </Page>
```

*Zdroj: Windows Dev Center: Blinky Sample [online]. 2015 [cit. 2015-11-21]. Dostupné z:
<http://msiot.github.io/content/en-US/win10/samples/Blinky.htm>*

Soubor MainPage.xaml.cs

```
// Copyright (c) Microsoft. All rights reserved.
using System; using
Windows.Devices.Gpio; using
```

```

Windows.UI.Xaml; using
Windows.UI.Xaml.Controls; using
Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Media;

namespace Blinky
{
    public sealed partial class MainPage : Page
    {
        private const int LED_PIN = 5;
        private GpioPin pin;           private
        GpioPinValue pinValue;         private
        DispatcherTimer timer;         private
        SolidColorBrush redBrush = new
        SolidColorBrush(Windows.UI.Colors.Red);
        private SolidColorBrush grayBrush = new
        SolidColorBrush(Windows.UI.Colors.LightGray);

        public MainPage()
        {
            InitializeComponent();

            timer = new DispatcherTimer();
            timer.Interval = TimeSpan.FromMilliseconds(500);
            timer.Tick +=
Timer_Tick;           InitGPIO();
if (pin != null)
{
    timer.Start();
}
private void InitGPIO()
{
    var gpio = GpioController.Default();
    // Show an error if there is no GPIO controller
if (gpio == null)
{
    pin = null;
    GpioStatus.Text = "Toto zařízení neobsahuje GPIO sběrnici.";
return;
}

    pin = gpio.OpenPin(LED_PIN);
    pinValue = GpioPinValue.High;
    pin.Write(pinValue);
    pin.SetDriveMode(GpioPinDriveMode.Output);

    GpioStatus.Text = "GPIO piny jsou nastaveny správně.";
}

private void Timer_Tick(object sender, object e)
{
    if (pinValue == GpioPinValue.High)
    {

```

```

        pinValue = GpioPinValue.Low;
pin.Write(pinValue);          LED.Fill
= grayBrush;
    }
else
    {
        pinValue = GpioPinValue.High;
pin.Write(pinValue);          LED.Fill
= redBrush;
    }
}
}
}
}

```

Upraveno: Windows Dev Center: Blinky Sample [online]. 2015 [cit. 2015-11-21]. Dostupné z: <http://msiot.github.io/content/en-US/win10/samples/Blinky.htm>

Soubor App.xaml.cs

```

// Copyright (c) Microsoft. All rights reserved.
using
System;
using
System.Collections.Generic;
using System.IO; using
System.Linq;
using
System.Runtime.InteropServices.WindowsRuntime;
using Windows.ApplicationModel; using
Windows.ApplicationModel.Activation; using
Windows.Foundation; using
Windows.Foundation.Collections; using
Windows.UI.Xaml; using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives; using
Windows.UI.Xaml.Data; using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media; using
Windows.UI.Xaml.Navigation;

// The Blank Application template is documented at
http://go.microsoft.com/fwlink/?LinkId=402347&clcid=0x409

namespace Blinky
{
    /// <summary>
    /// Provides application-specific behavior to supplement the default Application
class.
    /// </summary>
    sealed partial class App : Application
    {
        /// <summary>
        /// Initializes the singleton application object. This is the first line of
authored code
        /// executed, and as such is the logical equivalent of main() or WinMain().
        /// </summary>
public App()
    {
        InitializeComponent();
    }
}

```

```

        Suspending += OnSuspending;
    }

    /// <summary>
    /// Invoked when the application is launched normally by the end user.
    Other entry points
    /// will be used such as when the application is launched to open a specific
    file.
    /// </summary>
    /// <param name="e">Details about the launch request and process.</param>
    protected override void OnLaunched(LaunchActivatedEventArgs e)
    {
        #if
        DEBUG
            if (System.Diagnostics.Debugger.IsAttached)
            {
                DebugSettings.EnableFrameRateCounter = true;
            }
        #endif

        Frame rootFrame = Window.Current.Content as Frame;

        // Do not repeat app initialization when the Window already has content,
        // just ensure that the window is active
        if (rootFrame == null)
        {
            // Create a Frame to act as the navigation context and navigate to
            the first page
            rootFrame = new Frame();
            // Set the default language
            rootFrame.Language =
            Windows.Globalization.ApplicationLanguages.Languages[0];

            rootFrame.NavigationFailed += OnNavigationFailed;

            if (e.PreviousExecutionState ==
            ApplicationExecutionState.Terminated)
            {
                //TODO: Load state from previously suspended application

                // Place the frame in the current Window
                Window.Current.Content = rootFrame;
            }

            if (rootFrame.Content == null)
            {
                // When the navigation stack isn't restored navigate to the first
                page,
                // configuring the new page by passing required information as a
                navigation
                // parameter
                rootFrame.Navigate(typeof(MainPage), e.Arguments);
            }
        }
    }

```

```

        // Ensure the current window is active
        Window.Current.Activate();
    }

    /// <summary>
    /// Invoked when Navigation to a certain page fails
    /// </summary>
    /// <param name="sender">The Frame which failed navigation</param>
    /// <param name="e">Details about the navigation failure</param>
    void OnNavigationFailed(object sender, NavigationFailedEventArgs e)
    {
        throw new Exception("Failed to load Page " + e.SourcePageType.FullName);
    }

    /// <summary>
    /// Invoked when application execution is being suspended.  Application
    state is saved
    /// without knowing whether the application will be terminated or resumed
    with the contents
    /// of memory still intact.
    /// </summary>
    /// <param name="sender">The source of the suspend request.</param>
    /// <param name="e">Details about the suspend request.</param>
    private void OnSuspending(object sender, SuspendingEventArgs e)
    {
        var deferral = e.SuspendingOperation.GetDeferral();
        //TODO: Save application state and stop any background activity
        deferral.Complete();
    }
}

```

Zdroj: Automaticky generováno programem Visual Studio

Příloha E: Zdrojový kód informačního stánku

Soubor MainPage.xaml

```
<Page
    x:Class="IoTweb.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:local="using:IoTweb"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

        <Grid.RowDefinitions>
            <RowDefinition Height="65"></RowDefinition>
            <RowDefinition Height="*"></RowDefinition>
        </Grid.RowDefinitions>
        <!--Address bar-->
        <StackPanel Grid.Row="0" Orientation="Horizontal">
            <Button x:Name="button" Content="Domu" Height="61" Margin="0,2,0,0"
                VerticalAlignment="Top" Width="95" RenderTransformOrigin="0.711,0.511"
                FontFamily="Vladimir Script" FontSize="37.333" Background="#33020525"
                BorderBrush="#FF080E44" Foreground="#FFF31414" Click="button_Click"/>
        </StackPanel>

        <!--Web view control-->
        <WebView x:Name="webView" Grid.Row="1" />
    </Grid>

</Page>
```

</Page>

Zdroj: upraveno: Novaspirit: Coding Web Browser For Windows 10 IOT on Raspberry PI

2 [online]. 2015 [cit. 2015-12-08]. Dostupné z:

<http://www.novaspirit.com/2015/09/22/coding-web-browser-for-windows-10-iot-onraspberry-pi-2/>

Soubor MainPage.xml.cs

```
using System;
using
System.Collections.Generic;
using System.IO; using
System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Foundation; using
Windows.Foundation.Collections; using
Windows.UI.Popups; using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls; using
Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data; using
Windows.UI.Xaml.Input; using
```

```
Windows.UI.Xaml.Media; using
Windows.UI.Xaml.Navigation;
```

```
// Šablona položky prázdné stránky je popsána na adrese
http://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409.
```

```
namespace IoTweb
{
    /// <summary>
    /// Prázdná stránka, která se dá použít samostatně nebo v rámci objektu Frame
    /// </summary>
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
            webView.Navigate(new
Uri("http://www.centrumchodov.cz/W/do/centre/mapacentra"));
        }
        private async void button_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                webView.Navigate(new
Uri("http://www.centrumchodov.cz/W/do/centre/mapa-centra"));
            }
            catch (Exception f)
            {
                MessageBox dlg = new MessageBox("Error: " + f.Message);
                await dlg.ShowAsync();
            }
        }
    }
}
```

Zdroj: upraveno: Novaspirit: Coding Web Browser For Windows 10 IOT on Raspberry PI

2 [online]. 2015 [cit. 2015-12-08]. Dostupné z:
<http://www.novaspirit.com/2015/09/22/coding-web-browser-for-windows-10-iot-onraspberry-pi-2/>

Soubor App.xaml.cs

```
using System;
using
System.Collections.Generic;
using System.IO; using
System.Linq;
using
System.Runtime.InteropServices.WindowsRuntime;
using Windows.ApplicationModel; using
Windows.ApplicationModel.Activation; using
Windows.Foundation; using
Windows.Foundation.Collections; using
Windows.UI.Xaml; using Windows.UI.Xaml.Controls;
```

```

using Windows.UI.Xaml.Controls.Primitives; using
Windows.UI.Xaml.Data; using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media; using
Windows.UI.Xaml.Navigation;

namespace IoTweb
{
    /// <summary>
    /// Poskytuje chování specifické pro aplikaci, které doplňuje výchozí třídu
    Application.
    /// </summary>
    sealed partial class App : Application
    {
        /// <summary>
        /// Inicializuje objekt aplikace typu singleton. Jedná se o první řádek
        spuštění vytvořeného kódu,
        /// který je proto logickým ekvivalentem metod main() nebo WinMain().
        /// </summary>
        public App()
        {
            Microsoft.ApplicationInsights.WindowsAppInitializer.InitializeAsync(
                Microsoft.ApplicationInsights.WindowsCollectors.Metadata |
                Microsoft.ApplicationInsights.WindowsCollectors.Session);
            this.InitializeComponent();          this.Suspending += OnSuspending;
        }

        /// <summary>
        /// Vyvolá se při normálním spuštění aplikace koncovým uživatelem. Ostatní
        vstupní body
        /// se použijí například při spuštění aplikace za účelem
        otevření konkrétního souboru.          /// </summary>
        /// <param name="e">Podrobnosti o žádosti o spuštění a procesu</param>
        protected override void OnLaunched(LaunchActivatedEventArgs e)
        {
            #if
            DEBUG

                if (System.Diagnostics.Debugger.IsAttached)
                {
                    this.DebugSettings.EnableFrameRateCounter = true;
                }
            #endif

            Frame rootFrame = Window.Current.Content as Frame;
            // Neopakovat inicializaci aplikace, pokud už má okno obsah,
            // jenom ověřit, jestli je toto okno aktivní
            if (rootFrame == null)
            {
                // Vytvořit objekt Frame, který bude fungovat jako kontext navigace,
                a spustit procházení první stránky          rootFrame = new Frame();

                rootFrame.NavigationFailed += OnNavigationFailed;

                if (e.PreviousExecutionState ==
                ApplicationExecutionState.Terminated)
                {
                    //TODO: Načíst stav z dříve pozastavené aplikace

```

```

    }

    // Umístit rámec do aktuálního objektu Window
    Window.Current.Content = rootFrame;
}

if (rootFrame.Content == null)
{
    // Pokud není navigační zásobník obnovený, přejít na první stránku
    // a nakonfigurovat novou stránku předáním požadovaných informací ve formě
    // parametru navigace
    rootFrame.Navigate(typeof(MainPage), e.Arguments);
}
// Zkontrolovat, jestli je aktuální okno aktivní
Window.Current.Activate();
}

/// <summary>
/// Vyvolá se, když selže přechod na určitou stránku.
/// </summary>
/// <param name="sender">Objekt Frame, u kterého selhala navigace</param>
/// <param name="e">Podrobnosti o chybě navigace</param>
void OnNavigationFailed(object sender, NavigationFailedEventArgs e)
{
    throw new Exception("Failed to load Page " + e.SourcePageType.FullName);
}

/// <summary>
/// Vyvolá se při pozastavení běhu aplikace. Stav aplikace se uloží
/// bez informace, jestli se aplikace ukončí nebo obnoví s obsahem
/// stále neporušené paměti.
/// </summary>
/// <param name="sender">Zdroj žádosti o pozastavení</param>
/// <param name="e">Podrobnosti žádosti o pozastavení</param>
private void OnSuspending(object sender, SuspendingEventArgs e)
{
    var deferral = e.SuspendingOperation.GetDeferral();
    //TODO: Uložit stav aplikace a zastavit jakoukoliv aktivitu na pozadí
    deferral.Complete();
}
}
}

```

Zdroj: Automaticky generováno programem Visual Studio

Příloha F: Zdrojový kód webového prohlížeče

Soubor MainPage.xaml

```
<Page
    x:Class="IoTweb.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:local="using:IoTweb"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

        <Grid.RowDefinitions>
            <RowDefinition Height="65"></RowDefinition>
            <RowDefinition Height="*"></RowDefinition>
        </Grid.RowDefinitions>
        <!--Address bar-->
        <StackPanel Grid.Row="0" Orientation="Horizontal">
            <TextBox x:Name="Web_Address" FontSize="34.667" TextWrapping="Wrap"
                Text="http://www.jcu.cz" VerticalAlignment="Center"
                VerticalContentAlignment="Center" Height="54" Width="958" KeyUp="Web_Address_KeyUp"
                FontFamily="Vladimir Script" FontWeight="Bold"/>
            <Button x:Name="Go_Web" Content="Jdi" HorizontalAlignment="Right"
                VerticalAlignment="Center" Height="60" Width="107" Click="Go_Web_Click"
                FontFamily="Vladimir Script" FontSize="34.667" Foreground="#FF102774"/>
            <Button x:Name="button" Content="Domu" Height="61" Margin="0,2,0,0"
                VerticalAlignment="Top" Width="95" RenderTransformOrigin="0.711,0.511"
                FontFamily="Vladimir Script" FontSize="37.333" Background="#33020525"
                BorderBrush="#FF080E44" Foreground="#FFF31414" Click="button_Click"/>
        </StackPanel>

        <!--Web view control-->
        <WebView x:Name="webView" Grid.Row="1" />
    </Grid>

</Page>
```

Zdroj: *upraveno: Novaspirit: Coding Web Browser For Windows 10 IOT on Raspberry PI*

2 [online]. 2015 [cit. 2015-12-08]. Dostupné z:

<http://www.novaspirit.com/2015/09/22/coding-web-browser-for-windows-10-iot-onraspberry-pi-2/>

Soubor MainPage.xaml.cs

```
using System;
using
System.Collections.Generic;
using System.IO; using
System.Linq;
```

```

using
System.Runtime.InteropServices.WindowsRuntime;
using Windows.Foundation; using
Windows.Foundation.Collections; using
Windows.UI.Popups; using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls; using
Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data; using
Windows.UI.Xaml.Input; using
Windows.UI.Xaml.Media; using
Windows.UI.Xaml.Navigation;

```

// Šablona položky prázdné stránky je popsána na adrese
<http://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409>.

```

namespace IoTweb
{
    /// <summary>
    /// Prázdná stránka, která se dá použít samostatně nebo v rámci objektu Frame
    /// </summary>
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
            webView.Navigate(new Uri("http://jcu.cz"));
        }
        private void Go_Web_Click(object sender, RoutedEventArgs e)
        {
            DoWebNavigate();
        }
        private void Web_Address_KeyUp(object sender, KeyRoutedEventArgs e)
        {
            if (e.Key == Windows.System.VirtualKey.Enter)
            {
                DoWebNavigate();
            }
        }
        private async void DoWebNavigate()
        {
            try
            {
                if (Web_Address.Text.Length > 0)
                {
                    webView.Navigate(new Uri(Web_Address.Text));
                }
            }
            else
            {
                MessageDialog dlg = new MessageDialog("you need to enter a web address.");
                await dlg.ShowAsync();
            }
        }
        catch (Exception e)
        {
            MessageDialog dlg = new MessageDialog("Error: " + e.Message);
            await dlg.ShowAsync();
        }
    }
}

```

```

        }
    }
    private async void button_Click(object sender,
RoutedEventArgs e)
    {
        try
        {
            if (Web_Address.Text.Length > 0)
            {
                webView.Navigate(new Uri("http://jcu.cz"));

            }
        }
        else
        {
            MessageDialog dlg = new MessageDialog("Zadejte platnou webovou
adresu.");
            await dlg.ShowAsync();
        }
    }
    catch (Exception f)
    {
        MessageDialog dlg = new MessageDialog("Error: " + f.Message);
        await dlg.ShowAsync();
    }
}
}
}
}
}

```

Zdroj: upraveno: Novaspirit: Coding Web Browser For Windows 10 IOT on Raspberry PI

2 [online]. 2015 [cit. 2015-12-08]. Dostupné z:

<http://www.novaspirit.com/2015/09/22/coding-web-browser-for-windows-10-iot-onraspberry-pi-2/>

Soubor App.xaml.cs

```

using System;
using
System.Collections.Generic;
using System.IO; using
System.Linq;
using
System.Runtime.InteropServices.WindowsRuntime;
using Windows.ApplicationModel; using
Windows.ApplicationModel.Activation; using
Windows.Foundation; using
Windows.Foundation.Collections; using
Windows.UI.Xaml; using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives; using
Windows.UI.Xaml.Data; using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media; using
Windows.UI.Xaml.Navigation;

```

namespace IoTweb

```

{
    /// <summary>
    /// Poskytuje chování specifické pro aplikaci, které doplňuje výchozí třídu
    Application.
    /// </summary>
    sealed partial class App : Application
    {
        /// <summary>
        /// Inicializuje objekt aplikace typu singleton. Jedná se o první řádek
        spuštěného vytvořeného kódu,
        /// který je proto logickým ekvivalentem metod main() nebo WinMain().
        /// </summary>
        public App()
        {
            Microsoft.ApplicationInsights.WindowsAppInitializer.InitializeAsync(
                Microsoft.ApplicationInsights.WindowsCollectors.Metadata |
                Microsoft.ApplicationInsights.WindowsCollectors.Session);
            this.InitializeComponent();          this.Suspending += OnSuspending;
        }

        /// <summary>
        /// Vyvolá se při normálním spuštění aplikace koncovým uživatelem. Ostatní
        vstupní body
        /// se použijí například při spuštění aplikace za účelem
        otevření konkrétního souboru.          /// </summary>
        /// <param name="e">Podrobnosti o žádosti o spuštění a procesu</param>
        protected override void OnLaunched(LaunchActivatedEventArgs e)
        {
            #if DEBUG
                if (System.Diagnostics.Debugger.IsAttached)
                {
                    this.DebugSettings.EnableFrameRateCounter = true;
                }
            #endif

            Frame rootFrame = Window.Current.Content as Frame;

            // Neopakovat inicializaci aplikace, pokud už má okno obsah,
            // jenom ověřit, jestli je toto okno aktivní
            if (rootFrame == null)
            {
                // Vytvořit objekt Frame, který bude fungovat jako kontext navigace,
                a spustit procházení první stránky          rootFrame = new Frame();

                rootFrame.NavigationFailed += OnNavigationFailed;

                if (e.PreviousExecutionState ==
                ApplicationExecutionState.Terminated)
                {
                    //TODO: Načíst stav z dříve pozastavené aplikace
                }

                // Umístit rámeček do aktuálního objektu Window
                Window.Current.Content = rootFrame;
            }

```

```

        if (rootFrame.Content == null)
        {
            // Pokud není navigační zásobník obnovený, přejít na první stránku
            // a nakonfigurovat novou stránku předáním požadovaných informací ve formě
            // parametru navigace
            rootFrame.Navigate(typeof(MainPage), e.Arguments);
        }

        // Zkontrolovat, jestli je aktuální okno aktivní
        Window.Current.Activate();
    }

    /// <summary>
    /// Vvolá se, když selže přechod na určitou stránku.
    /// </summary>
    /// <param name="sender">Objekt Frame, u kterého selhala navigace</param>
    /// <param name="e">Podrobnosti o chybě navigace</param>
    void OnNavigationFailed(object sender, NavigationFailedEventArgs e)
    {
        throw new Exception("Failed to load Page " + e.SourcePageType.FullName);
    }

    /// <summary>
    /// Vvolá se při pozastavení běhu aplikace. Stav aplikace se uloží
    /// bez informace, jestli se aplikace ukončí nebo obnoví s obsahem
    /// stále neporušené paměti.
    /// </summary>
    /// <param name="sender">Zdroj žádosti o pozastavení</param>
    /// <param name="e">Podrobnosti žádosti o pozastavení</param>
    private void OnSuspending(object sender, SuspendingEventArgs e)
    {
        var deferral = e.SuspendingOperation.GetDeferral();
        //TODO: Uložit stav aplikace a zastavit jakoukoliv aktivitu na pozadí
        deferral.Complete();
    }
}
}

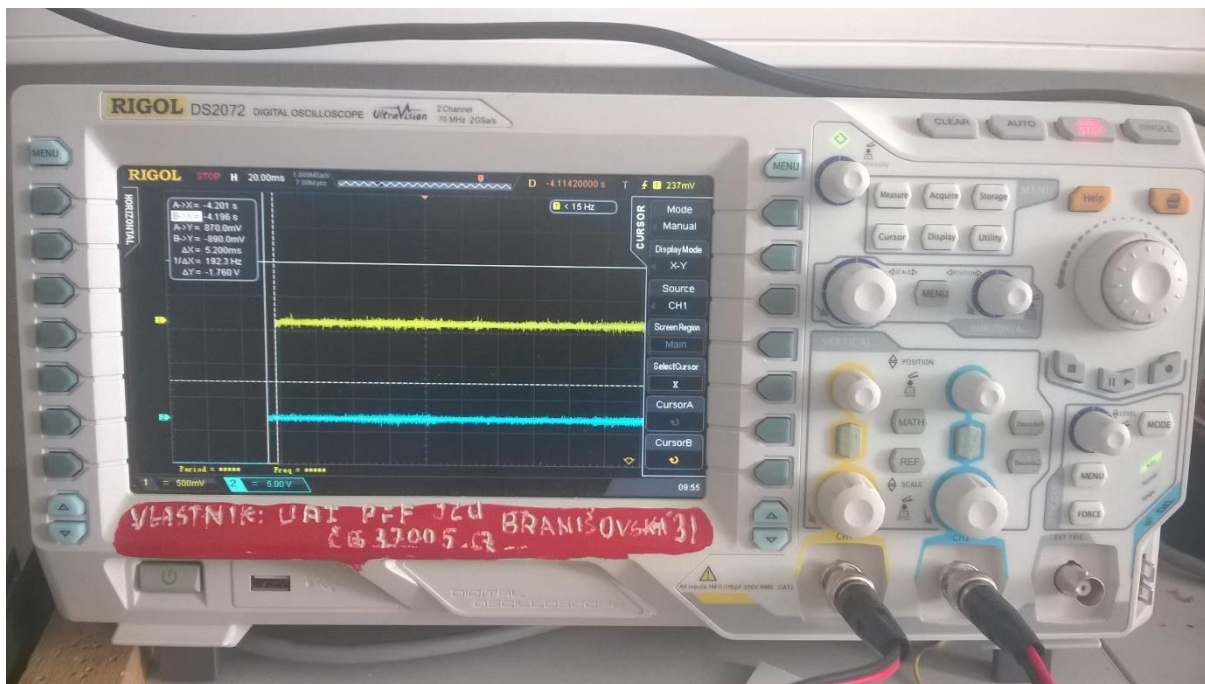
```

Zdroj: Automaticky generováno programem Visual Studio

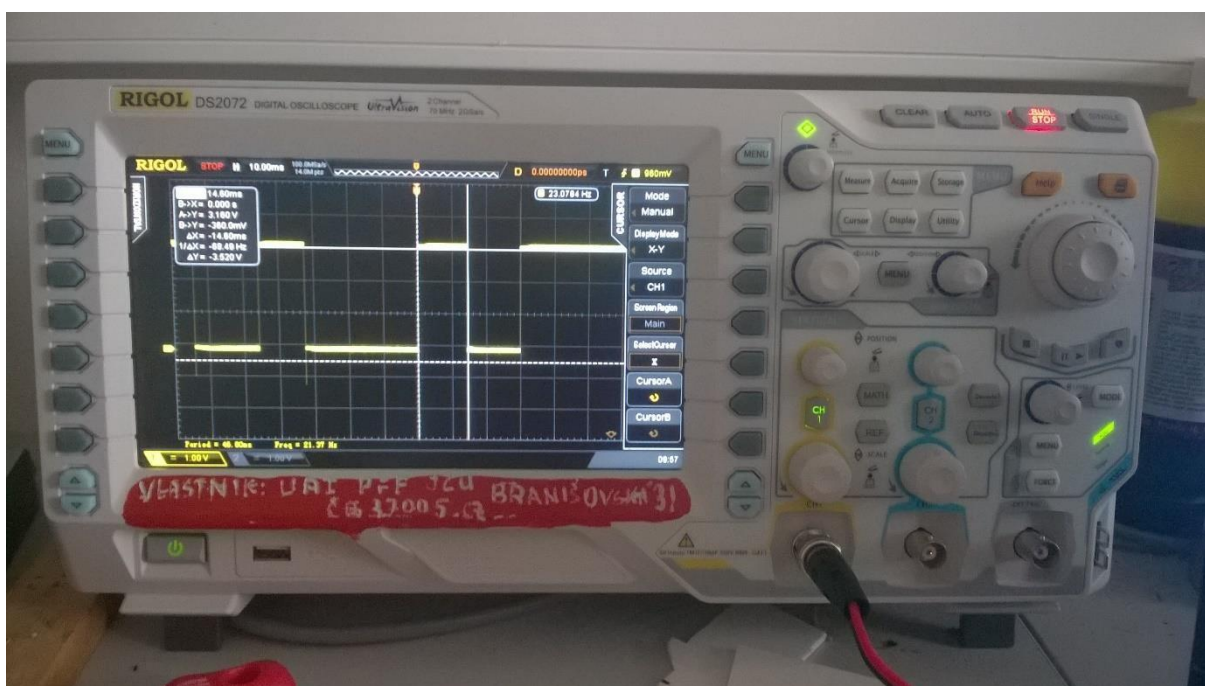
Příloha G: Zdrojový kód měření reakční doby GPIO sběrnice pro Raspbian

```
from gpiozero import LED,  
Button from signal import pause  
led = LED(23) button =  
Button(24) led.on()  
button.when_released = led.on  
button.when_pressed = led.off  
pause()
```

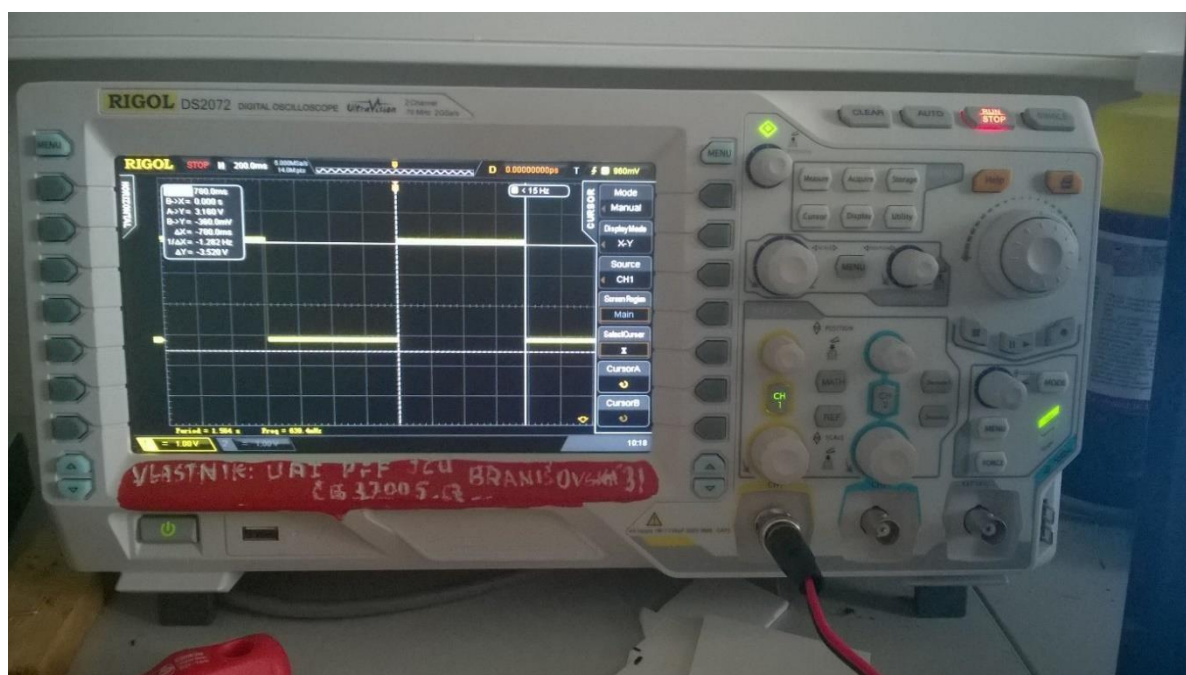
Příloha H: Obrazová dokumentace měření na GPIO sběrnici



Měření reakční doby události GPIO

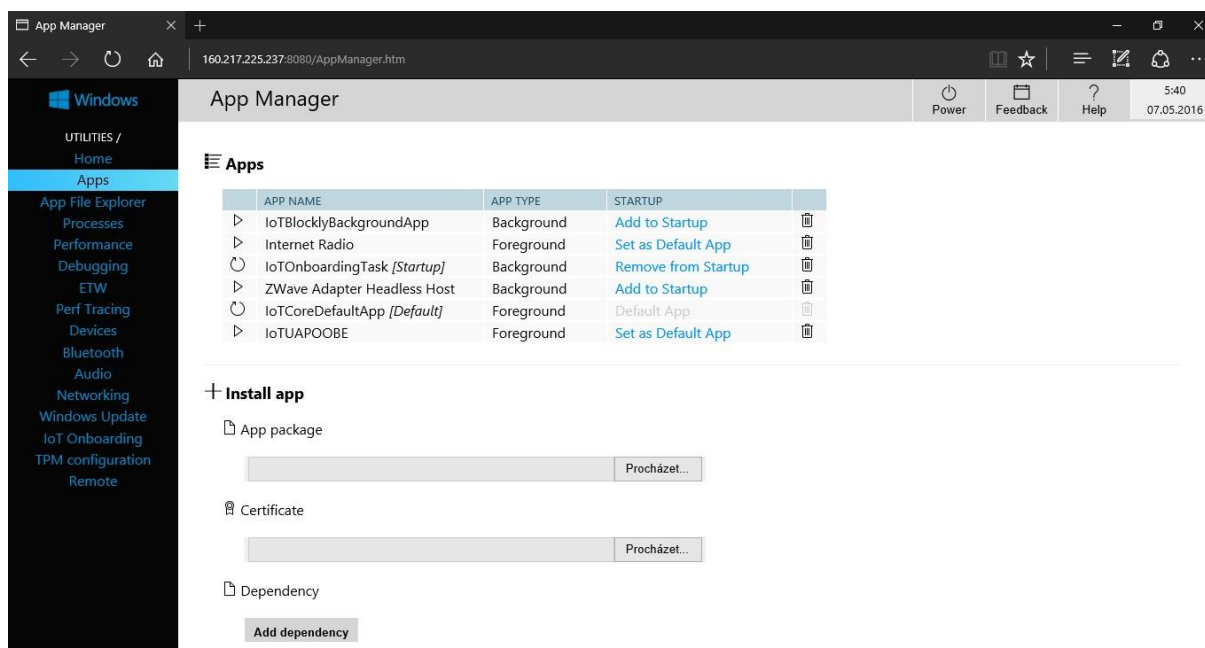


Výstupní signál z SW časovače Pi2

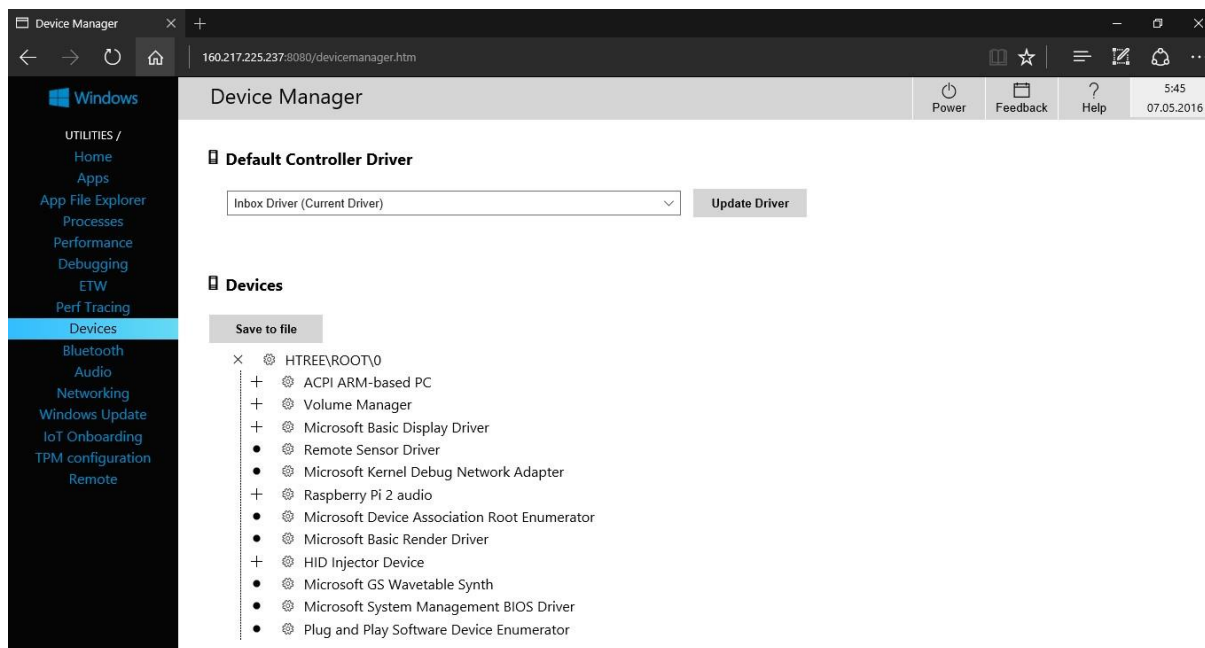


Výstupní signál z SW časovače Pi2

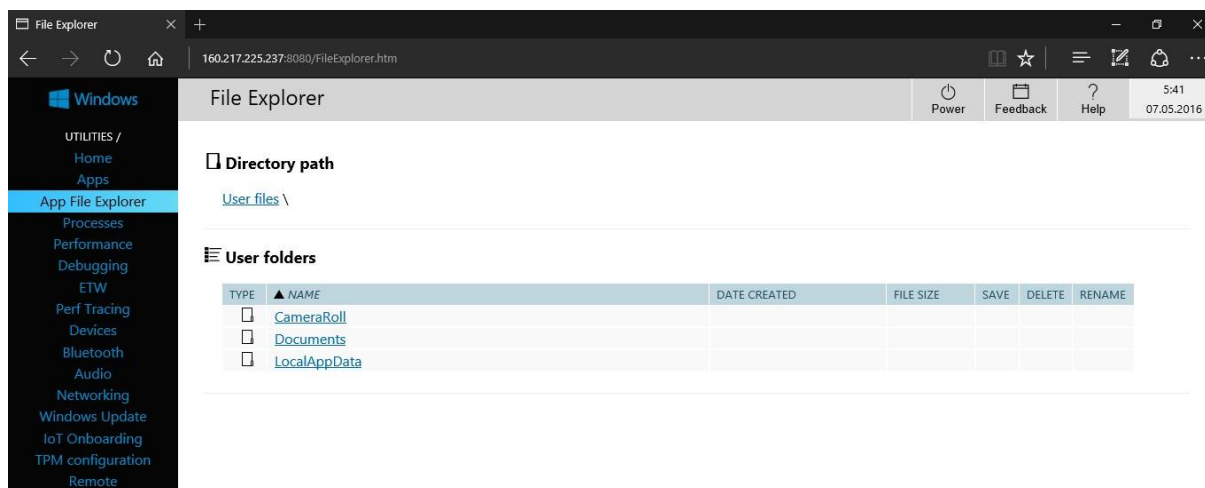
Příloha I: Obrázky novinek ve webovém rozhraní pro správu OS



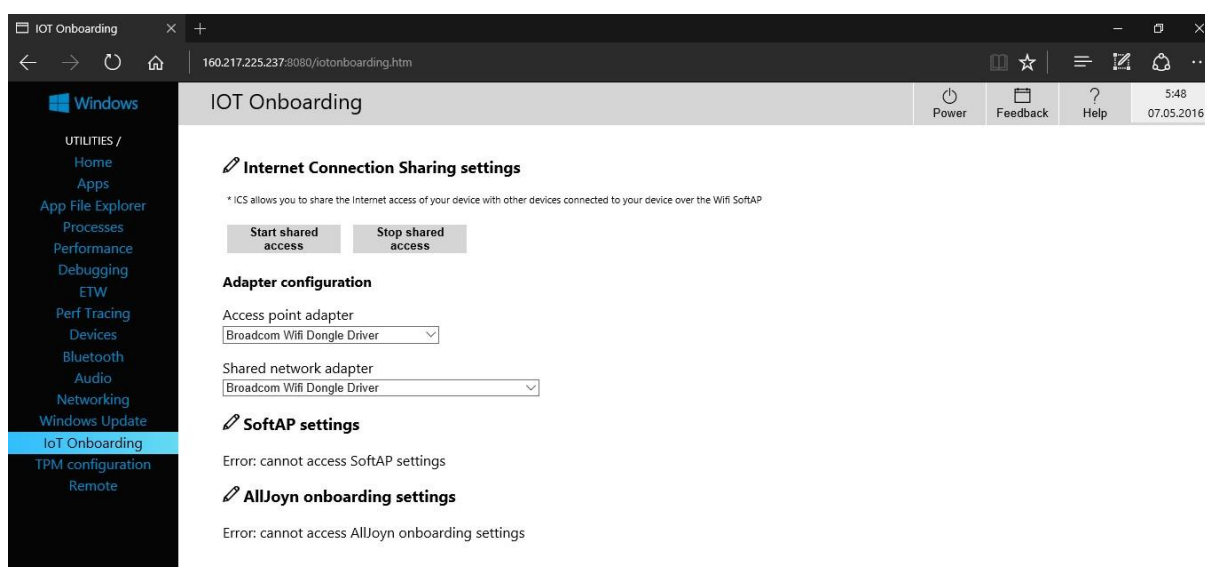
Nová zpráva programů pomocí webového rozhraní



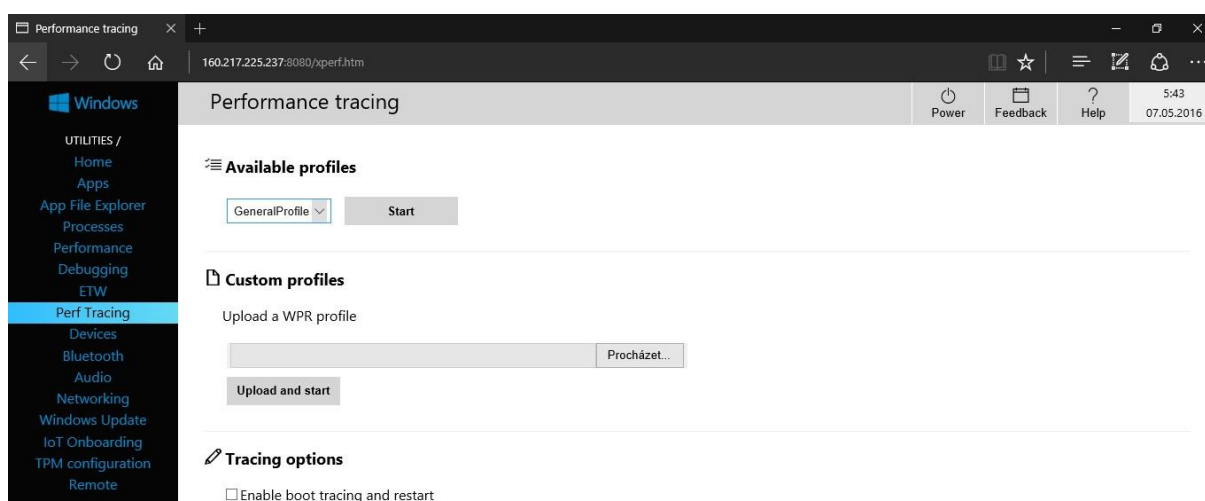
Nový Device Manager



Průzkumník souborů



Sdílení síťového připojení

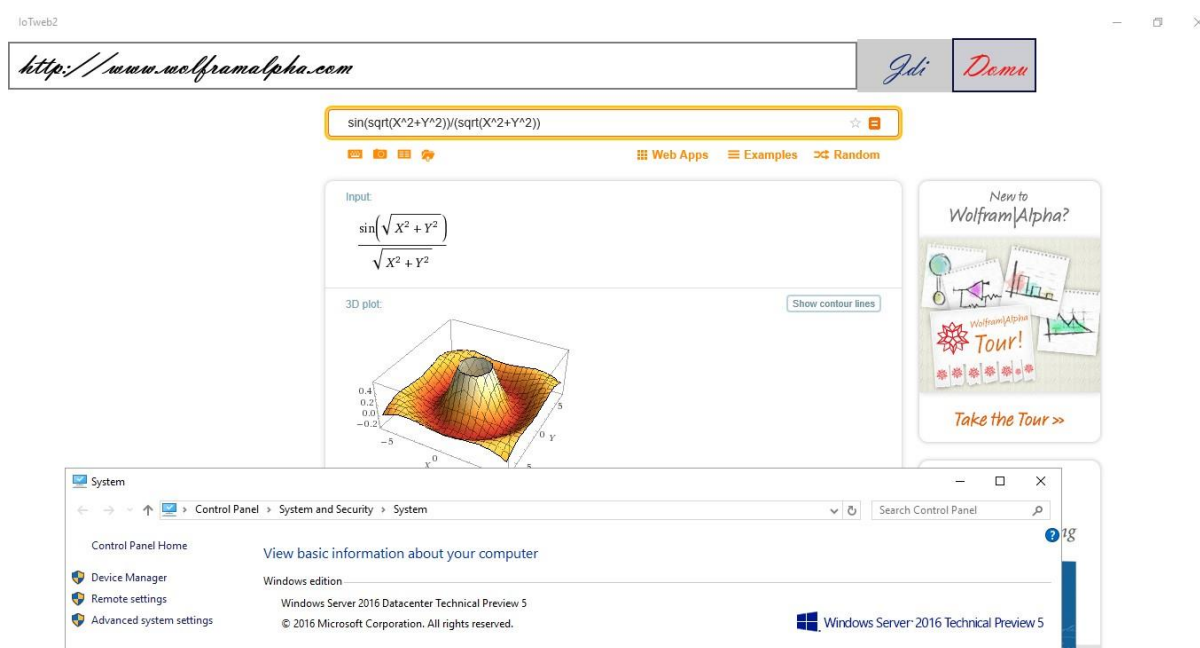


Performance Tracing

Příloha J: Obrázky testu práce s internetem



Přehrávání hudby z Youtube pomocí Windows 10 IoT Core a Pi2



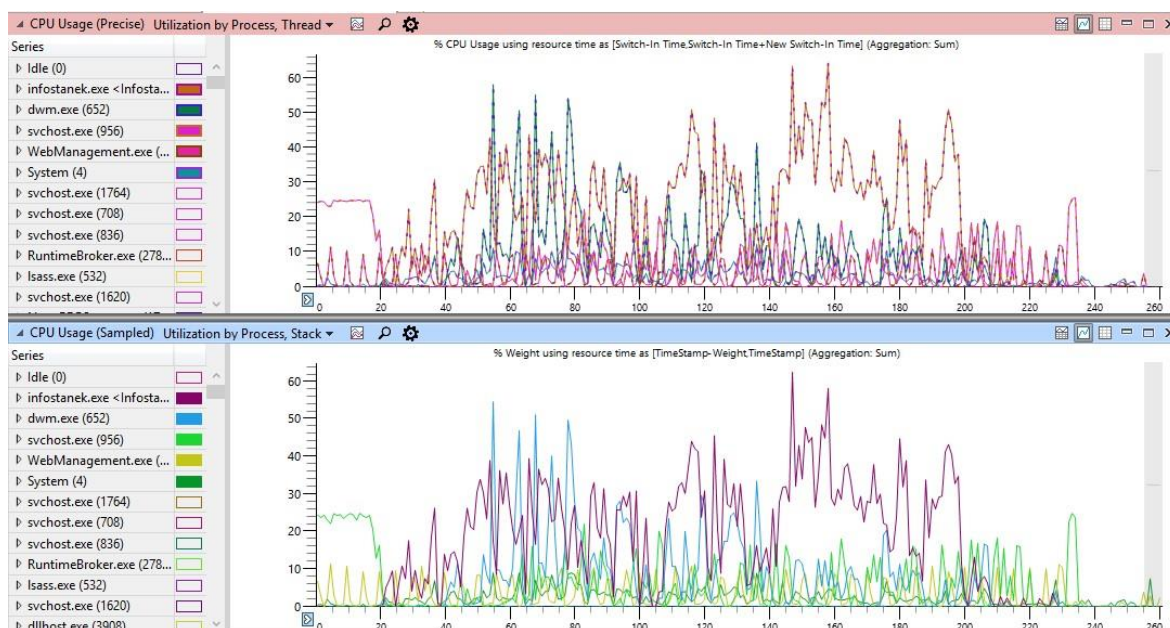
Webový prohlížeč pro Windows 10 IoT Core spuštěný na Windows Server 2016 TP5

Příloha K: Postup stažení systému pro Pi3

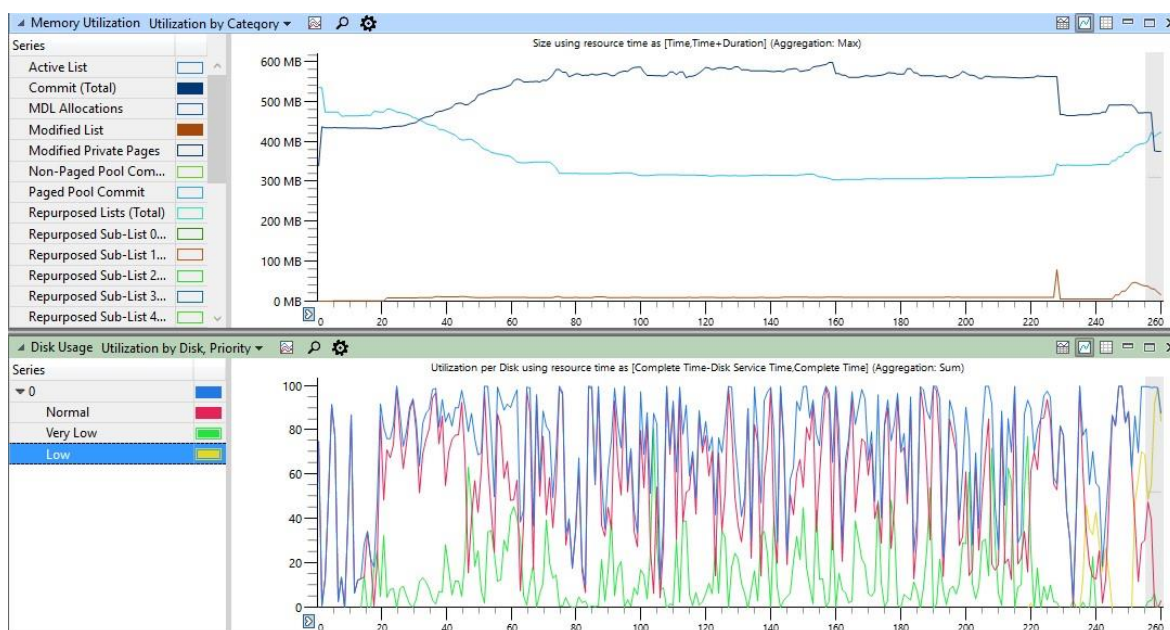
The screenshot shows the Microsoft website for downloading the Windows 10 IoT Core Insider Preview. On the left is a 'Create an account' form with fields for First name, Last name, User name (pre-filled with 'someone@example.com'), and Password. On the right, under the 'Download Windows 10 IoT Core Insider Preview' header, there are instructions and a 'Download Now' button. A 'Select the edition' dropdown is set to 'Windows 10 IoT Core Insider Preview', and a 'Select the device' dropdown is set to 'Raspberry Pi'. Both dropdowns have 'Confirm' buttons. A 'Go Back' button is at the top left of the form area, and a 'Reload' button is below it. Arrows indicate the flow: from 'Go Back' to the account form, from 'Confirm' under 'Select the edition' to the 'Download Now' button, and from 'Confirm' under 'Select the device' to the 'Download Now' button. The 'Download Now' button is highlighted in blue.

Část postupu registrace a stažení Insider Buildu Windows 10 IoT. Zdroj: *Setup with NOOBS: Using a Raspberry Pi 3*: [online]. Redmnod, 2016 [cit. 2016-07-19]. Dostupné z: <https://developer.microsoft.com/cscz/windows/iot/win10/Noobs.htm>

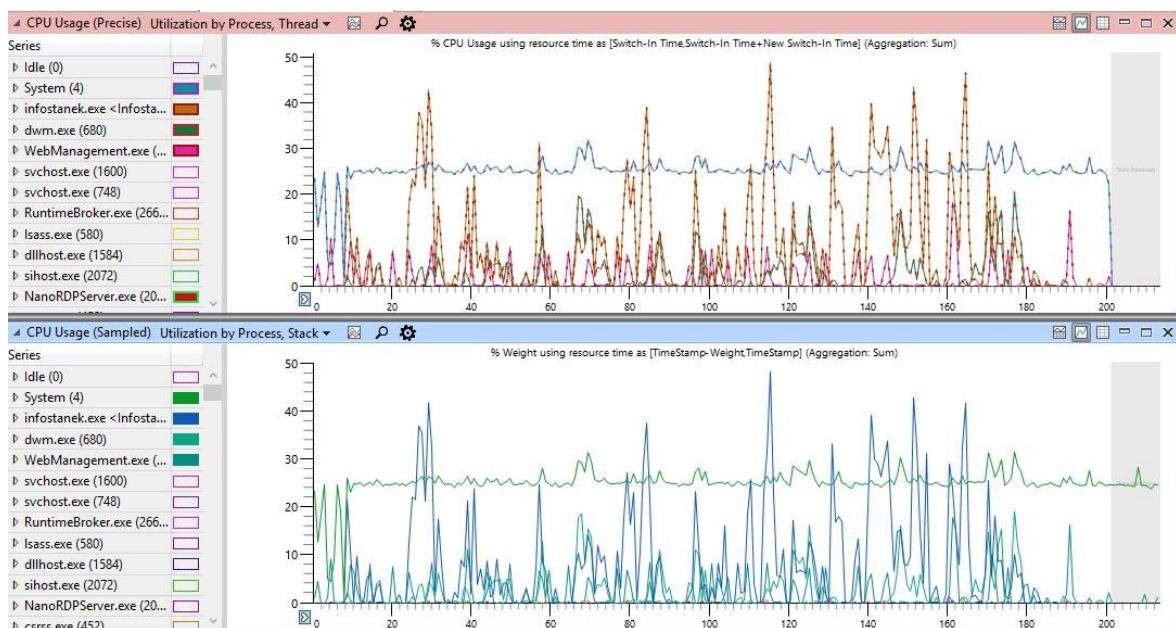
Příloha L: Grafy vytížení systému při simulaci informačního stánku



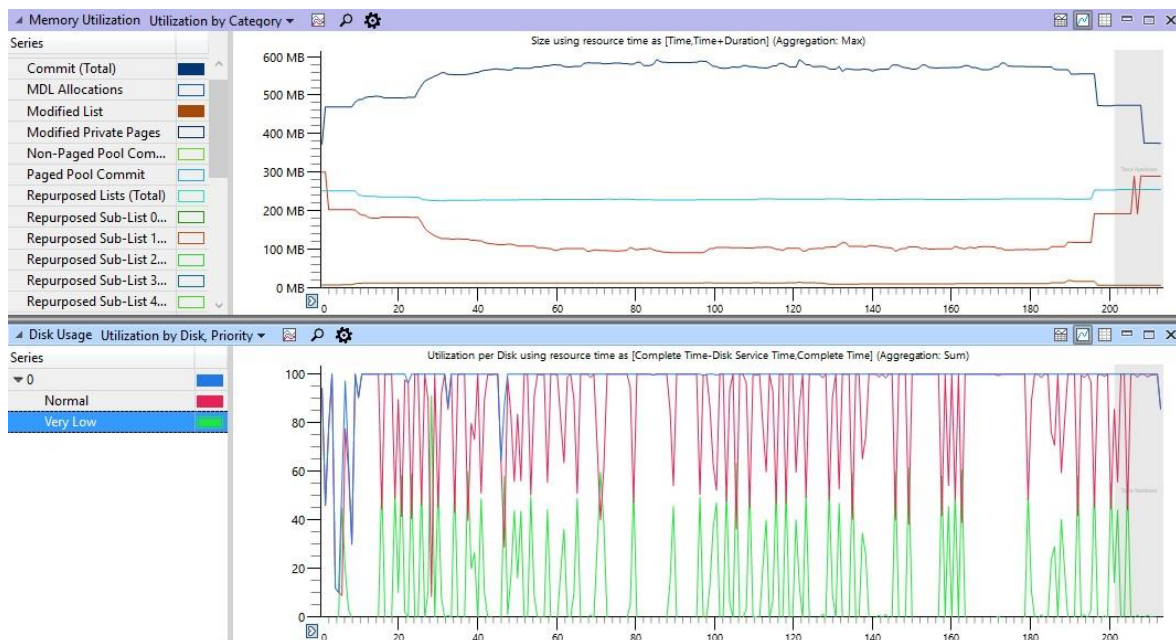
CPU trace z běhu aplikace teploměr na Pi2



Memory a Disk trace z běhu aplikace Teploměr na Pi2



CPU trace z běhu aplikace teploměr na Pi3

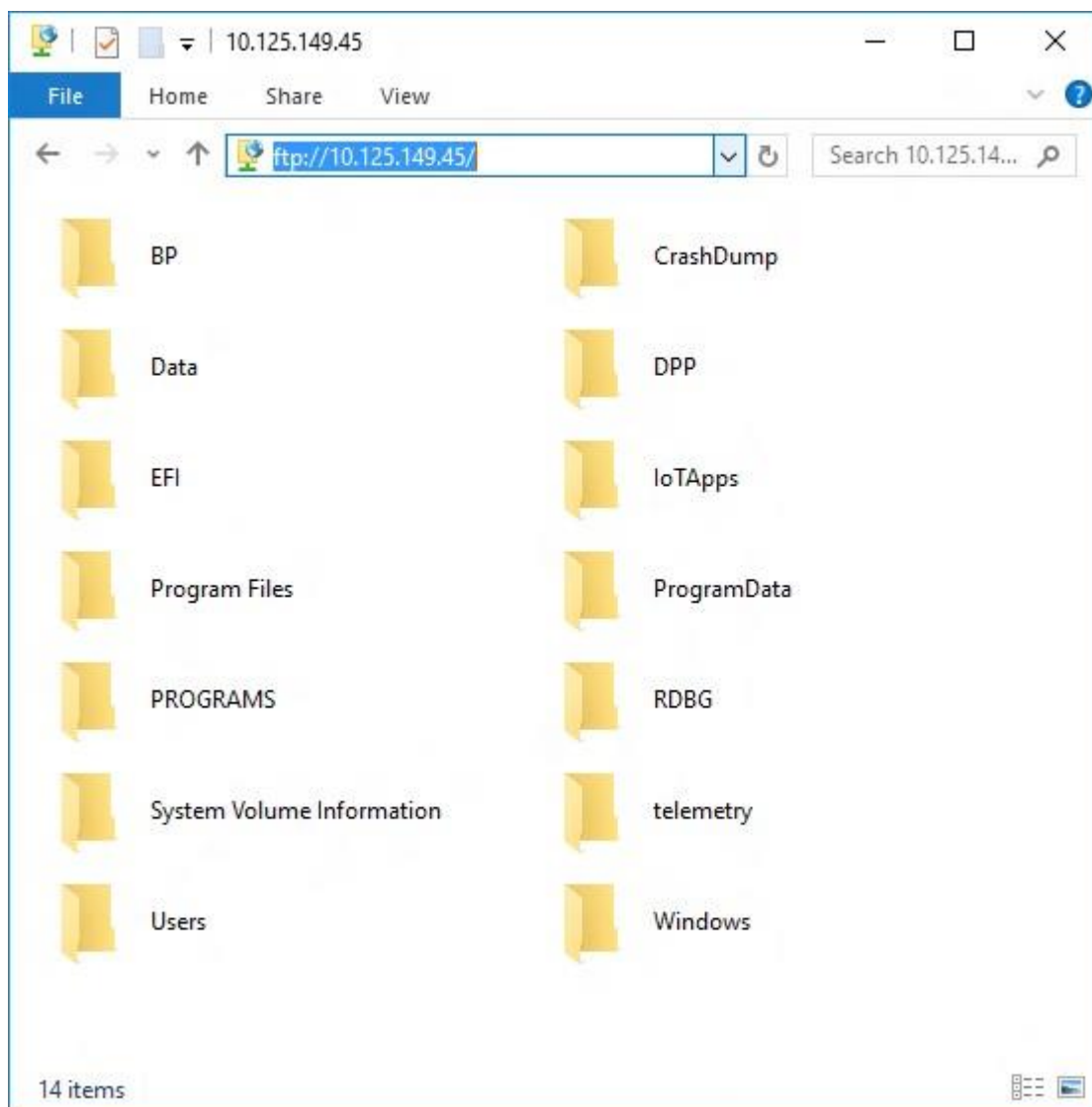


Memory a Disk trace z běhu aplikace Teploměr na Pi3

Příloha M: Práce s FTP

Pro jednorázové spuštění FTP potřebujeme použít PowerShell, kde zadáme příkaz:
`start C:\Windows\System32\ftpd.exe` Pokud si nejsme zcela jisti, zda se nám podařilo FTP úspěšně spustit, tak si to ověříme pomocí příkazu: `tlset`

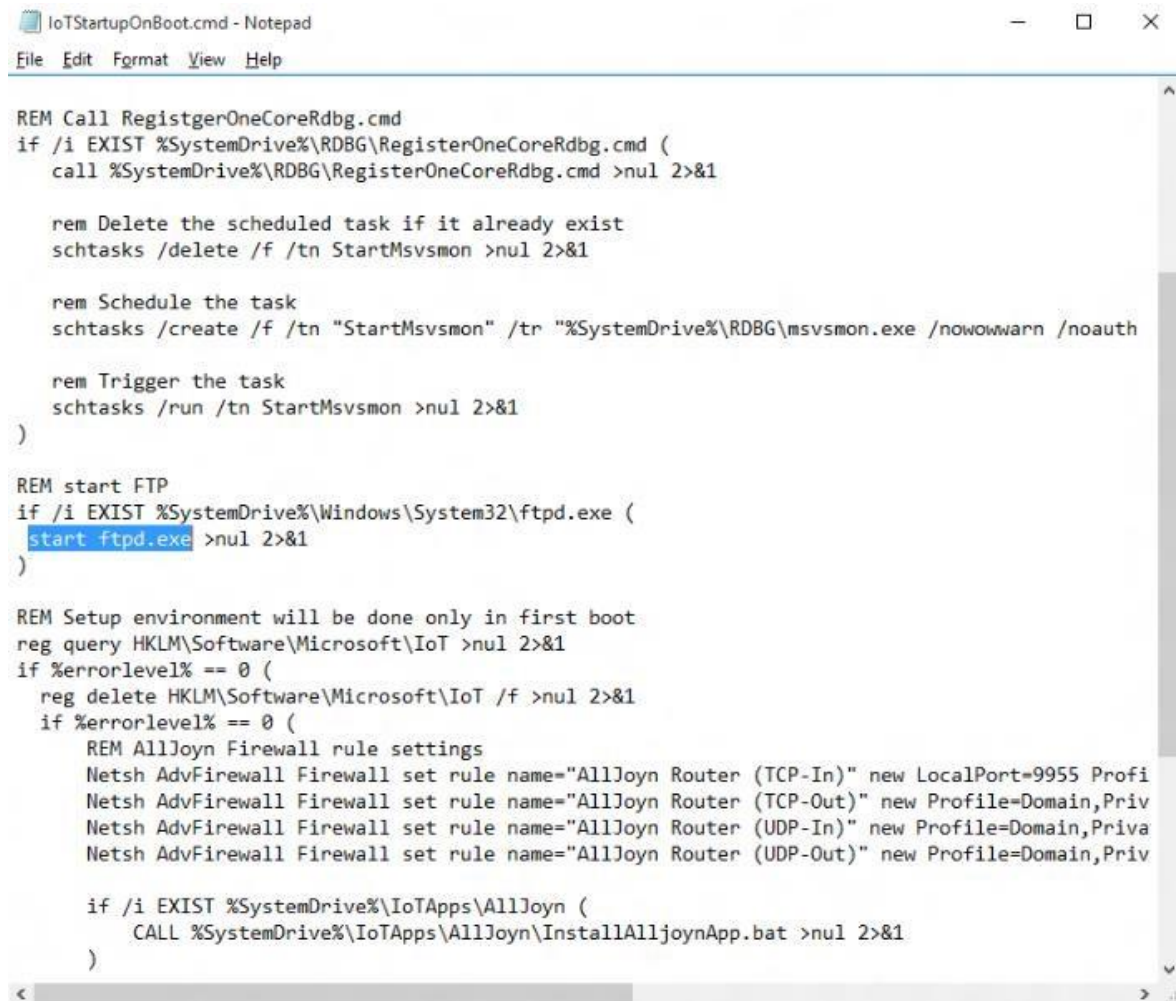
Nyní se již můžeme pomocí FTP klienta, nebo Průzkumníku Windows připojit k FTP našeho Windows IoT Core a pracovat se soubory.



Ukázkový obsah disku zpřístupněný pomocí FTP. Zdroj: File Transfer Protocol. Windows Dev Center [online]. Redmond: Microsoft, 2016 [cit. 2016-11-05]. Dostupné z: <https://developer.microsoft.com/enus/windows/iot/docs/ftp>

Pro opětovné vypnutí FTP slouží v PowerShellu příkaz: `kill -processname ftpd*`

Pokud chceme FTP spouštět se startem systému, musíme upravit soubor IoTStartupOnBoot.cmd dle následujícího obrázku:



```
REM Call RegistgerOneCoreRdbg.cmd
if /i EXIST %SystemDrive%\RDBG\RegisterOneCoreRdbg.cmd (
    call %SystemDrive%\RDBG\RegisterOneCoreRdbg.cmd >nul 2>&1

    rem Delete the scheduled task if it already exist
    schtasks /delete /f /tn StartMsvsmon >nul 2>&1

    rem Schedule the task
    schtasks /create /f /tn "StartMsvsmon" /tr "%SystemDrive%\RDBG\msvsmon.exe /nowowarn /noauth

    rem Trigger the task
    schtasks /run /tn StartMsvsmon >nul 2>&1
)

REM start FTP
if /i EXIST %SystemDrive%\Windows\System32\ftpd.exe (
    start ftpd.exe >nul 2>&1
)

REM Setup environment will be done only in first boot
reg query HKLM\Software\Microsoft\IoT >nul 2>&1
if %errorlevel% == 0 (
    reg delete HKLM\Software\Microsoft\IoT /f >nul 2>&1
    if %errorlevel% == 0 (
        REM AllJoyn Firewall rule settings
        Netsh AdvFirewall Firewall set rule name="AllJoyn Router (TCP-In)" new LocalPort=9955 Profi
        Netsh AdvFirewall Firewall set rule name="AllJoyn Router (TCP-Out)" new Profile=Domain,Priv
        Netsh AdvFirewall Firewall set rule name="AllJoyn Router (UDP-In)" new Profile=Domain,Priva
        Netsh AdvFirewall Firewall set rule name="AllJoyn Router (UDP-Out)" new Profile=Domain,Priv

        if /i EXIST %SystemDrive%\IoTApps\AllJoyn (
            CALL %SystemDrive%\IoTApps\AllJoyn\InstallAllJoynApp.bat >nul 2>&1
        )
    )
)
```

Úprava souboru *IoTStartupOnBoot.cmd* pro automatické spouštění FTP. Zdroj: File Transfer Protocol. Windows Dev Center [online]. Redmond: Micorosoft, 2016 [cit. 2016-11-05]. Dostupné z: <https://developer.microsoft.com/en-us/windows/iot/docs/ftp>